IEEE SA
STANDARDS ASSOCIATION

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

Base Specifications, Issue 8

STANDARDS

IEEE Computer Society

and

The Open Group

Developed by the
Microprocessor Standards Committee

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

The Open Group Standard Base Specifications, Issue 8

IEEE

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

The Open Group Standard, Base Specifications, Issue 8

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

Base Specifications, Issue 8

Developed by the

Microprocessor Committee
of the
IEEE Computer Society

and

The Open Group

Approved 20 May 2024

IEEE SA Standards Board

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

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

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

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

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

Published 14 June 2024 by IEEE in the United States of America.
PDF: ISBN 979-8-8557-0793-9 STD26978
Print: ISBN 979-8-8557-0794-6 STDPD26978

Published 14 June 2024 by The Open Group in the United Kingdom
Doc. Number: C243
ISBN: 1-957866-40-6

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

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

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

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

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

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

The Open Group

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

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

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

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

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

Important Notices and Disclaimers Concerning IEEE Standards Documents

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

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

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

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

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

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

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

Translations

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

Use by artificial intelligence systems

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

Official statements

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

Comments on standards

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

Laws and regulations

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

Data privacy

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

Copyrights

Photocopies

Updating of IEEE Standards documents

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

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

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

Errata

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

[1] Available at: https://ieeexplore.ieee.org/browse/standards/collection/ieee.
[2] Available at: https://standards.ieee.org/standard/index.html.

Patents

IEEE standards are developed in compliance with the IEEE SA Patent Policy.[3]

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

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

IMPORTANT NOTICE

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

[3] Available at: https://standards.ieee.org/about/sasb/patcom/materials.html.

Participants

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

The Austin Group

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

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

Austin Group Technical Reviewers

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

Austin Group Working Group Members

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

The Open Group

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

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

Base Working Group Members

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

IEEE

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

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

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

Boon Chong Ang
Steven Bezner
Diego Chiozzi
Donald W. Cragun
Andrew Fieldsend
David Fuschi
Jie Guan
Joseph M. Gwinn
Werner Hoelzl
Andrew Josey
Piotr Karocki
Kenneth Lang
Rajesh Murthy
Venkatesha Prasad
Stephen Schwarm
Walter Struppler
Oren Yuen
Janusz Zalewski

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

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

Sara R. Biyabani
Ted Burse
Stephen Dukes
Doug Edwards
J. Travis Griffith
Guido R. Hiertz
Ronald W Hotchkiss
Hao Hu
Yousef Kimiagar
Joseph L. Koepfinger*
Howard Li
Xiaohui Liu
John Haiying Lu
Kevin W. Lu
Hiroshi Mano
Paul Nikolich
Robby Robson
Lei Wang
F. Keith Waters
Sha Wei
Philip B. Winston
Don Wright

*Member Emeritus

Introduction

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

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

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

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

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

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

Background

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

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

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

[4] The Austin Group is named after the location of the inaugural meeting held at the IBM facility in Austin, Texas in September 1998.

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

Audience

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

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

Purpose

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

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

— Systems running on a broad range of hardware

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

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

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

  — By standardizing an interface like one in an historical implementation; for example, directories

  — By specifying an interface that is readily implementable in terms of, and backwards-compatible with, historical implementations, such as the extended tar format defined in the pax utility

  — By specifying an interface that, when added to an historical implementation, will not conflict with it; for example, the sigaction() function

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

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

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

POSIX.1-2024

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

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

Base Definitions

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

This volume is structured as follows:

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

Comprehensive references are available in the Index.

System Interfaces

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

This volume is structured as follows:

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

Comprehensive references are available in the Index.

Shell and Utilities

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

This volume is structured as follows:

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

Comprehensive references are available in the Index.

Rationale (Informative)

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

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

Within this volume, the following terms are used:

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

Typographical Conventions

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

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

Reference	Example	Notes
C-Language Data Structure	**aiocb**	

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

Note that:

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

2. Unless otherwise noted, the quotes shall not be used as input or output. When used in a list item, the quotes are omitted. The literal characters <apostrophe> (also known as single-quote) and <backslash> are either shown as the C constants `'\''` and `'\\'`, respectively, or as the special characters <apostrophe>, single-quote, and <backslash> depending on context.

3. The style selected for some of the special characters, such as <newline>, matches the form of the input given to the *localedef* utility. Generally, the characters selected for this special treatment are those that are not visually distinct, such as the control characters <tab> or <newline>.

4. Names surrounded by braces represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C **#define** construct, or obtained at runtime from functions or utilities that return limit or configuration values.

5. Brackets shown in this font, `"[ ]"`, are part of the syntax and do not indicate optional items. In syntax the `'|'` symbol is used to separate alternatives, and ellipses (`"..."`) are used to show that additional arguments are optional.

Shading is used to identify extensions and options.

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

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

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

1. (*a*,*b*) means the range of all values from *a* to *b*, including neither *a* nor *b*.

2. [*a*,*b*] means the range of all values from *a* to *b*, including *a* and *b*.

3. [*a*,*b*) means the range of all values from *a* to *b*, including *a*, but not *b*.

4. (*a*,*b*] means the range of all values from *a* to *b*, including *b*, but not *a*.

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

Contents

List of Figures

List of Tables

Trademarks

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

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

AIX® and IBM® are registered trademarks of International Business Machines Corporation.

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

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

BSD™ is a trademark of the University of California, Berkeley, USA.

Hewlett Packard®, HP®, and HP-UX® are registered trademarks of HP Hewlett Packard Group LLC.

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

Linux® is a registered trademark of Linus Torvalds.

Sun® and Sun Microsystems® are registered trademarks of Oracle America, Inc.

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

Acknowledgements

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

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

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

Referenced Documents

Normative References

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

Informative References

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

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

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

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

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

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

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

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

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

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

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

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

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

IEEE Std 754™-1985
IEEE Std 754-1985 (Reaff 1990), IEEE Standard for Binary Floating-Point Arithmetic.

IEEE Std 854™-1987
IEEE Std 854-1987, IEEE Standard for Radix-Independent Floating-Point Arithmetic.

IEEE Std 1003.9™-1992
IEEE Std 1003.9-1992, IEEE Standard for Information Technology — POSIX FORTRAN 77 Language Interfaces — Part 1: Binding for System Application Program Interface API.

IETF RFC 791
Internet Protocol, Version 4 (IPv4), September 1981 (available at: www.ietf.org/rfc/rfc0791.txt).

IETF RFC 819
The Domain Naming Convention for Internet User Applications, Z. Su, J. Postel, August 1982 (available at: www.ietf.org/rfc/rfc0819.txt).

IETF RFC 919
Broadcasting Internet Datagrams, J. Mogul, October 1984 (available at: www.ietf.org/rfc/rfc0919.txt).

IETF RFC 920
Domain Requirements, J. Postel, J. Reynolds, October 1984 (available at: www.ietf.org/rfc/rfc0920.txt).

IETF RFC 921
Domain Name System Implementation Schedule, J. Postel, October 1984 (available at: www.ietf.org/rfc/rfc0921.txt).

IETF RFC 922
Broadcasting Internet Datagrams in the Presence of Subnets, J. Mogul, October 1984 (available at: www.ietf.org/rfc/rfc0922.txt).

IETF RFC 1034
Domain Names — Concepts and Facilities, P. Mockapetris, November 1987 (available at: www.ietf.org/rfc/rfc1034.txt).

IETF RFC 1035
Domain Names — Implementation and Specification, P. Mockapetris, November 1987 (available at: www.ietf.org/rfc/rfc1035.txt).

IETF RFC 1123
Requirements for Internet Hosts — Application and Support, R. Braden, October 1989 (available at: www.ietf.org/rfc/rfc1123.txt).

IETF RFC 1951
DEFLATE Compressed Data Format Specification version 1.3, P. Deutsch, May 1996 (available at: www.ietf.org/rfc/rfc1951.txt).

IETF RFC 1952
GZIP file format specification version 4.3, P. Deutsch, May 1996 (available at: www.ietf.org/rfc/rfc1952.txt).

IETF RFC 2045
Multipurpose Internet Mail Extensions (MIME), Part 1: Format of Internet Message Bodies, N. Freed, N. Borenstein, November 1996 (available at: www.ietf.org/rfc/rfc2045.txt).

IETF RFC 2181
Clarifications to the DNS Specification, R. Elz, R. Bush, July 1997 (available at: www.ietf.org/rfc/rfc2181.txt).

IETF RFC 3596
DNS Extensions to Support IP Version 6, S. Thomson, C. Huitema, V. Ksinant, M. Souissi, October 2003 (available at: www.ietf.org/rfc/rfc3596.txt).

IETF RFC 4291
IP Version 6 Addressing Architecture, R. Hinden, S. Deering, February 2006 (available at: www.ietf.org/rfc/rfc4291.txt).

IETF RFC 5322
Internet Message Format, P. Resnick, October 2008 (available at: www.ietf.org/rfc/rfc5322.txt).

IETF RFC 6557
Procedures for Maintaining the Time Zone Database, E. Lear, P. Eggert, February 2012 (available at: www.ietf.org/rfc/rfc6557.txt).

IETF RFC 8200
Internet Protocol, Version 6 (IPv6) Specification, S. Deering, R. Hinden, July 2017 (available at: www.ietf.org/rfc/rfc8200.txt).

Internationalisation Guide
Guide, July 1993, Internationalisation Guide, Version 2 (ISBN: 1-859120-02-4, G304), published by The Open Group.

ISO 2375: 1985
ISO 2375: 1985, Data Processing — Procedure for Registration of Escape Sequences.

ISO 8652: 1987
ISO 8652: 1987, Programming Languages — Ada (technically identical to ANSI standard 1815A-1983).

ISO/IEC 1539: 1991
ISO/IEC 1539: 1991, Information Technology — Programming Languages — Fortran (technically identical to the ANSI X3.9-1978 standard [FORTRAN 77]).

ISO/IEC 4873: 1991
ISO/IEC 4873: 1991, Information Technology — ISO 8-bit Code for Information Interchange — Structure and Rules for Implementation.

ISO/IEC 6429: 1992
ISO/IEC 6429: 1992, Information Technology — Control Functions for Coded Character Sets.

ISO/IEC 6937: 1994
ISO/IEC 6937: 1994, Information Technology — Coded Graphic Character Set for Text Communication — Latin Alphabet.

ISO/IEC 8802-3: 1996
ISO/IEC 8802-3: 1996, Information Technology — Telecommunications and Information Exchange Between Systems — Local and Metropolitan Area Networks — Specific Requirements — Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications.

ISO/IEC 8859
ISO/IEC 8859, Information Technology — 8-Bit Single-Byte Coded Graphic Character Sets:

Part 1: Latin Alphabet No. 1
Part 2: Latin Alphabet No. 2
Part 3: Latin Alphabet No. 3
Part 4: Latin Alphabet No. 4
Part 5: Latin/Cyrillic Alphabet
Part 6: Latin/Arabic Alphabet
Part 7: Latin/Greek Alphabet
Part 8: Latin/Hebrew Alphabet
Part 9: Latin Alphabet No. 5
Part 10: Latin Alphabet No. 6
Part 11: Latin/Thai Alphabet
Part 13: Latin Alphabet No. 7
Part 14: Latin Alphabet No. 8 (Celtic)
Part 15: Latin Alphabet No. 9
Part 16: Latin Alphabet No. 10

ISO/IEC 9899: 1990
ISO/IEC 9899: 1990, Programming Languages — C, including Amendment 1: 1995 (E), C Integrity (Multibyte Support Extensions (MSE) for ISO C).

ISO POSIX-1: 1996
ISO/IEC 9945-1: 1996, Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language] (identical to ANSI/IEEE Std 1003.1-1996). Incorporating ANSI/IEEE Stds 1003.1-1990, 1003.1b-1993, 1003.1c-1995, and 1003.1i-1995.

ISO POSIX-2: 1993
ISO/IEC 9945-2: 1993, Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities (identical to ANSI/IEEE Std 1003.2™-1992, as amended by ANSI/IEEE Std 1003.2a™-1992).

Issue 1
X/Open Portability Guide, July 1985 (ISBN: 0-444-87839-4).

Issue 2
X/Open Portability Guide, January 1987:

- Volume 1: XVS Commands and Utilities (ISBN: 0-444-70174-5)
- Volume 2: XVS System Calls and Libraries (ISBN: 0-444-70175-3)

Issue 3
X/Open Specification, 1988, 1989, February 1992:

- Commands and Utilities, Issue 3 (ISBN: 1-872630-36-7, C211); this specification was formerly X/Open Portability Guide, Issue 3, Volume 1, January 1989, XSI Commands and Utilities (ISBN: 0-13-685835-X, XO/XPG/89/002)
- System Interfaces and Headers, Issue 3 (ISBN: 1-872630-37-5, C212); this specification was formerly X/Open Portability Guide, Issue 3, Volume 2, January 1989, XSI System Interface and Headers (ISBN: 0-13-685843-0, XO/XPG/89/003)
- Curses Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapters 9 to 14 inclusive; this specification was formerly X/Open Portability Guide, Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)

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

Issue 4

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

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

Issue 4, Version 2

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

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

Issue 5

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

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

Issue 6

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

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

Knuth Article

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

KornShell

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

MSE Working Draft

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

POSIX.0:1995

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

POSIX.1:1988

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

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

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

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

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

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

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

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

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

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

POSIX.13: 1998
IEEE Std 1003.13™-1998, IEEE Standard for Information Technology — Standardized Application Environment Profile (AEP) — POSIX Realtime Application Support.

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

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

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

SVID, Issue 1
American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 1; Morristown, NJ, UNIX Press, 1985.

SVID, Issue 2
American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 2; Morristown, NJ, UNIX Press, 1986.

SVID, Issue 3
American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 3; Morristown, NJ, UNIX Press, 1989.

The AWK Programming Language
Aho, Alfred V., Kernighan, Brian W., and Weinberger, Peter J., *The AWK Programming Language*, Reading, MA, Addison-Wesley 1988.

The C Programming Language
Kernighan, Brian W. and Ritchie, Dennis M., *The C Programming Language*, Englewood Cliffs, NJ, Prentice Hall, 1st Edition (February 1978) ISBN 0-13-110163-3; 2nd Edition (March 1988) ISBN 0-13-110362-8.

UNIX Programmer's Manual
American Telephone and Telegraph Company, *UNIX Time-Sharing System: UNIX Programmer's Manual*, 7th Edition, Murray Hill, NJ, Bell Telephone Laboratories, January 1979.

XNS, Issue 4
CAE Specification, August 1994, Networking Services, Issue 4 (ISBN: 1-85912-049-0, C438), published by The Open Group.

XNS, Issue 5
CAE Specification, February 1997, Networking Services, Issue 5 (ISBN: 1-85912-165-9, C523), published by The Open Group.

XNS, Issue 5.2
Technical Standard, January 2000, Networking Services (XNS), Issue 5.2 (ISBN: 1-85912-241-8, C808), published by The Open Group.

X/Open Curses, Issue 4, Version 2
CAE Specification, May 1996, X/Open Curses, Issue 4, Version 2 (ISBN: 1-85912-171-3, C610), published by The Open Group.

Yacc
Yacc: Yet Another Compiler Compiler, Stephen C. Johnson, 1978.

Source Documents

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

AIX 3.2 Manual
AIX Version 3.2 For RISC System/6000, Technical Reference: Base Operating System and Extensions, 1990, 1992 (Part No. SC23-2382-00).

OSF/1
OSF/1 Programmer's Reference, Release 1.2 (ISBN: 0-13-020579-6).

OSF AES
Application Environment Specification (AES) Operating System Programming Interfaces Volume, Revision A (ISBN: 0-13-043522-8).

System V Release 2.0

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

— UNIX System V Release 2.0 Programming Guide (April 1984 - Issue 2).

System V Release 4.2

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

The Open Group Standard

Vol. 1:

Base Definitions, Issue 8

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

Chapter 1
Introduction

1.1 Scope

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

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

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

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

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

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

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

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

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

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

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

1.2 Word Usage

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall* equals *is required to*).[1, 2]

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

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

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

1.3 Conformance

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

---

1. The use of the word *must* is deprecated and cannot be used when stating mandatory requirements; *must* is used only to describe unavoidable situations.
2. The use of *will* is deprecated and cannot be used when stating mandatory requirements; *will* is only used in statements of fact.

1.4 Normative References

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

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

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

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

ISO C (C17)
: ISO/IEC 9899: 2018, Programming Languages — C.

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

1.5 Change History

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

1.6 Terminology

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.7 Definitions and Concepts

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

1.8 Portability

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

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

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

1.8.1 Codes

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

ADV Advisory Information
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

CD C-Language Development Utilities
The functionality described is optional.

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

CPT Process CPU-Time Clocks
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

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

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

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

DC Device Control
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

FR FORTRAN Runtime Utilities
The functionality described is optional.

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

FSC File Synchronization
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

IP6 IPV6
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the IP6 margin legend.

MC1 Non-Robust Mutex Priority Protection or Non-Robust Mutex Priority Inheritance or Robust Mutex Priority Protection or Robust Mutex Priority Inheritance
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

This is a shorthand notation for combinations of multiple option codes.

Where applicable, functions are marked with the MC1 margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the MC1 margin legend.

Refer to Section 1.8.2 (on page 12).

ML Process Memory Locking
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

MLR Range Memory Locking
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

MSG **Message Passing**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

MX **IEC 60559 Floating-Point**
The functionality described is optional. The functionality described is mandated by the ISO C standard only for implementations that define __STDC_IEC_559__.

MXC **IEC 60559 Complex Floating-Point**
The functionality described is optional. The functionality described is mandated by the ISO C standard only for implementations that define __STDC_IEC_559_COMPLEX__.

MXX **IEC 60559 Floating-Point Extension**
The functionality described is optional. The functionality described is part of the IEC 60559 Floating-Point option, but is an extension to the ISO C standard.

OB **Obsolescent**
The functionality described may be removed in a future version of this volume of POSIX.1-2024. Strictly Conforming POSIX Applications and Strictly Conforming XSI Applications shall not use obsolescent features.

Where applicable, the material is identified by use of the OB margin legend.

OF **Output Format Incompletely Specified**
The functionality described is an XSI extension. The format of the output produced by the utility is not fully specified. It is therefore not possible to post-process this output in a consistent fashion. Typical problems include unknown length of strings and unspecified field delimiters.

Where applicable, the material is identified by use of the OF margin legend.

OH **Optional Header**
In the SYNOPSIS section of some interfaces in the System Interfaces volume of POSIX.1-2024 an included header is marked as in the following example:

OH
```
#include <sys/types.h>
#include <fcntl.h>
int open(const char *path, int oflag, ...);
```

The OH margin legend indicates that the optional header defines constants that will be needed if the function is called with certain flag arguments; thus it may be required for some of the functionality described, but is not needed otherwise.

PIO **Prioritized Input and Output**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

PS **Process Scheduling**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

margin legend.

RPI Robust Mutex Priority Inheritance
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

RPP Robust Mutex Priority Protection
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

RS Raw Sockets
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

SD Software Development Utilities
The functionality described is optional.

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

SHM Shared Memory Objects
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

SIO Synchronized Input and Output
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

SPN Spawn
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

SS Process Sporadic Server
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

TCT Thread CPU-Time Clocks

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

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

TPI Non-Robust Mutex Priority Inheritance

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

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

TPP Non-Robust Mutex Priority Protection

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

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

TPS Thread Execution Scheduling

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

Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TPS margin legend.

TSA Thread Stack Address Attribute

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

Where applicable, functions are marked with the TSA margin legend for the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TSA margin legend.

TSH Thread Process-Shared Synchronization

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

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

TSP Thread Sporadic Server

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

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

TSS Thread Stack Size Attribute
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

TYM Typed Memory Objects
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

UP User Portability Utilities
The functionality described is optional.

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

UU UUCP Utilities
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

XSI X/Open System Interfaces
The functionality described is part of the X/Open Systems Interfaces option. Functionality marked XSI is an extension to the ISO C standard. Application developers may confidently make use of such extensions on all systems supporting the X/Open System Interfaces option.

If an entire SYNOPSIS section is shaded and marked XSI, all the functionality described in that reference page is an extension. See Section 2.1.4 (on page 19).

1.8.2 Margin Code Notation

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

A Feature Dependent on One or Two Options

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

SHM This feature requires support for only the Shared Memory Objects option.

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

A Feature Dependent on Either of the Options Denoted

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

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

A Feature Dependent on More than Two Options

The following shorthand notations are used:

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

Large Sections Dependent on an Option

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

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

Chapter 2
Conformance

2.1 Implementation Conformance

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

2.1.1 Requirements

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

1. The system shall support all utilities, functions, and facilities defined within POSIX.1-2024 that are required for POSIX conformance (see Section 2.1.3, on page 17). These interfaces shall support the functional behavior described herein.
2. The system may support the X/Open System Interfaces (XSI) option as described in Section 2.1.4 (on page 19).
3. The system may support one or more options as described under Section 2.1.5 (on page 20). When an implementation claims that an option is supported, all of its constituent parts shall be provided.
4. The system may provide non-standard extensions. These are features not required by POSIX.1-2024 and may include, but are not limited to:

   — Additional functions

   — Additional headers

   — Additional symbols in standard headers

   — Additional utilities

   — Additional options for standard utilities

   — Additional environment variables

   — Additional file types

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

   — Dynamically populated file systems (for example, **/proc**)

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

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

459 Section 2.2.1, on page 27).

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

464 ### 2.1.2 Documentation

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

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

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

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

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

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

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

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

2.1.3 POSIX Conformance

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

2.1.3.1 *POSIX System Interfaces*

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

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

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

  — _POSIX_CHOWN_RESTRICTED

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

  — _POSIX_NO_TRUNC

    Pathname components longer than {NAME_MAX} generate an error.

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

  — Symbolic constants defined with the value 202405L:

    _POSIX_ASYNCHRONOUS_IO
    _POSIX_BARRIERS
    _POSIX_CLOCK_SELECTION
    _POSIX_MAPPED_FILES
    _POSIX_MEMORY_PROTECTION
    _POSIX_MONOTONIC_CLOCK
    _POSIX_READER_WRITER_LOCKS
    _POSIX_REALTIME_SIGNALS
    _POSIX_SEMAPHORES
    _POSIX_SPIN_LOCKS
    _POSIX_THREAD_SAFE_FUNCTIONS
    _POSIX_THREADS
    _POSIX_TIMEOUTS
    _POSIX_TIMERS
    _POSIX2_C_BIND

  — Symbolic constants defined with a value greater than zero:

    _POSIX_JOB_CONTROL
    _POSIX_REGEXP
    _POSIX_SAVED_IDS
    _POSIX_SHELL

— Symbolic constants defined with a value other than –1.

_POSIX_VDISABLE

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

- The system may support one or more options (see Section 2.1.6, on page 25) denoted by the following symbolic constants:

  _POSIX_ADVISORY_INFO
  _POSIX_CPUTIME
  _POSIX_DEVICE_CONTROL
  _POSIX_FSYNC
  _POSIX_IPV6
  _POSIX_MEMLOCK
  _POSIX_MEMLOCK_RANGE
  _POSIX_MESSAGE_PASSING
  _POSIX_PRIORITIZED_IO
  _POSIX_PRIORITY_SCHEDULING
  _POSIX_RAW_SOCKETS
  _POSIX_SHARED_MEMORY_OBJECTS
  _POSIX_SPAWN
  _POSIX_SPORADIC_SERVER
  _POSIX_SYNCHRONIZED_IO
  _POSIX_THREAD_ATTR_STACKADDR
  _POSIX_THREAD_CPUTIME
  _POSIX_THREAD_ATTR_STACKSIZE
  _POSIX_THREAD_PRIO_INHERIT
  _POSIX_THREAD_PRIO_PROTECT
  _POSIX_THREAD_PRIORITY_SCHEDULING
  _POSIX_THREAD_PROCESS_SHARED
  _POSIX_THREAD_SPORADIC_SERVER
  _POSIX_TYPED_MEMORY_OBJECTS
  _XOPEN_CRYPT
  _XOPEN_REALTIME
  _XOPEN_REALTIME_THREADS
  _XOPEN_UNIX

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

2.1.3.2 *POSIX Shell and Utilities*

The following requirements apply to the shell and utilities:

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

583 POSIX2_C_DEV
584 POSIX2_CHAR_TERM
585 POSIX2_FORT_RUN
586 POSIX2_LOCALEDEF
587 POSIX2_SW_DEV
588 POSIX2_UPE
589 XOPEN_UNIX
590 XOPEN_UUCP

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

596 ### 2.1.4 XSI Conformance

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

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

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

606 #### *2.1.4.1 XSI System Interfaces*

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

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

611 • The system shall support the following options defined within POSIX.1-2024 (see Section
612 2.1.6, on page 25):

613 _POSIX_FSYNC
614 _POSIX_THREAD_ATTR_STACKADDR
615 _POSIX_THREAD_ATTR_STACKSIZE
616 _POSIX_THREAD_PROCESS_SHARED

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

619 — Encryption

620 — Realtime

621 — Advanced Realtime

— Realtime Threads

— Advanced Realtime Threads

2.1.4.2 *XSI Shell and Utilities Conformance*

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

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

admin	*delta*	*rmdel*	*val*
cflow	*get*	*sact*	*what*
ctags	*nm*	*sccs*	
cxref	*prs*	*unget*	

2.1.5 Option Groups

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

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

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

2.1.5.1 *Subprofiling Considerations*

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

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

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

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

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

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

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

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

  — If the limit is specified as having a fixed value, it shall not be changed by a profile.

  — If a limit is specified as having a minimum or maximum acceptable value, it may be changed by a profile as follows:

    — A profile may increase a minimum acceptable value, but shall not make a minimum acceptable value smaller.

    — A profile may reduce a maximum acceptable value, but shall not make a maximum acceptable value larger.
- A profile shall not change a limit specified as having a minimum or maximum value into a limit specified as having a fixed value.
- A profile shall not create new limits.
- Any implementation that conforms to POSIX.1-2024 (including all options and extended limits required by the profile) shall also conform to that profile, except for the possible omission from **<unistd.h>** of a profile support indicator macro required by the profile.

---

4. As an example, the File System profiling option group provides underlying support for pathname resolution and file creation which are needed by any interface in POSIX.1-2024 that parses a *path* argument. If a profile requires support for the Device Input and Output profiling option group but does not require support for the File System profiling option group, the profile needs to specify how pathname resolution is to behave in that profile, how the O_CREAT flag to *open*( ) is to be handled (and the use of the character `'a'` in the *mode* argument of *fopen*( ) when a pathname argument names a file that does not exist), and specify lots of other details.

5. As an example, POSIX.1-2024 requires that implementations claiming to support the Range Memory Locking option also support the Process Memory Locking option. A profile could require that the Range Memory Locking option had to be supplied without requiring that the Process Memory Locking option be supplied as long as the profile specifies everything an application developer or system implementor would have to know to build an application or implementation conforming to the profile.

6. Note that the profile could just specify that any use of the features not specified by the profile would produce undefined or unspecified results.

2.1.5.2 XSI Option Groups

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

The following Option Groups are defined.

Encryption

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

OB *crypt*( ), *encrypt*( ), *setkey*( )

These functions are marked CRYPT.

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

An implementation that claims conformance to this Option Group shall set _XOPEN_CRYPT to a value other than –1.

Realtime

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

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

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

An implementation that claims conformance to this Option Group shall set _XOPEN_REALTIME to a value other than –1.

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

_POSIX_FSYNC
_POSIX_MEMLOCK
_POSIX_MEMLOCK_RANGE
_POSIX_MESSAGE_PASSING
_POSIX_PRIORITIZED_IO
_POSIX_PRIORITY_SCHEDULING
_POSIX_SHARED_MEMORY_OBJECTS
_POSIX_SYNCHRONIZED_IO

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

742 _POSIX_MEMLOCK
743 _POSIX_MEMLOCK_RANGE
744 _POSIX_MESSAGE_PASSING
745 _POSIX_PRIORITY_SCHEDULING
746 _POSIX_SHARED_MEMORY_OBJECTS
747 _POSIX_SYNCHRONIZED_IO

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

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

757 **Advanced Realtime**

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

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

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

766 _POSIX_ADVISORY_INFO
767 _POSIX_CPUTIME
768 _POSIX_SPAWN
769 _POSIX_SPORADIC_SERVER
770 _POSIX_TYPED_MEMORY_OBJECTS

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

773 _POSIX_ADVISORY_INFO
774 _POSIX_CPUTIME
775 _POSIX_SPAWN
776 _POSIX_SPORADIC_SERVER
777 _POSIX_TYPED_MEMORY_OBJECTS

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

Realtime Threads

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

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

_POSIX_THREAD_PRIO_INHERIT
_POSIX_THREAD_PRIO_PROTECT
_POSIX_THREAD_PRIORITY_SCHEDULING
_POSIX_THREAD_ROBUST_PRIO_INHERIT
_POSIX_THREAD_ROBUST_PRIO_PROTECT

Where applicable, whole pages are marked REALTIME THREADS, together with the appropriate option margin legend for the SYNOPSIS section (see Section 1.8.1, on page 7).

An implementation that claims conformance to this Option Group shall set _XOPEN_REALTIME_THREADS to a value other than –1.

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

_POSIX_THREAD_PRIO_INHERIT
_POSIX_THREAD_PRIO_PROTECT
_POSIX_THREAD_PRIORITY_SCHEDULING
_POSIX_THREAD_ROBUST_PRIO_INHERIT
_POSIX_THREAD_ROBUST_PRIO_PROTECT

Advanced Realtime Threads

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

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

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

_POSIX_THREAD_CPUTIME
_POSIX_THREAD_SPORADIC_SERVER

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

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

_POSIX_THREAD_CPUTIME
_POSIX_THREAD_SPORADIC_SERVER

2.1.6 Options

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

1. Option not supported for compilation.

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

2. Option always supported.

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

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

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

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

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

2.1.6.1 *System Interfaces*

Refer to **<unistd.h>**, Constants for Options and Option Groups (on page 458) for the list of options.

862 *2.1.6.2 Shell and Utilities*

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

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

866 CD POSIX2_C_DEV
867 The system supports the C-Language Development Utilities option.

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

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

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

878 POSIX2_CHAR_TERM
879 The system supports the Terminal Characteristics option. This value need not be present on
880 a system not supporting the User Portability Utilities option.

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

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

885 FR POSIX2_FORT_RUN
886 The system supports the FORTRAN Runtime Utilities option.

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

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

890 POSIX2_LOCALEDEF
891 The system supports the Locale Creation Utilities option.

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

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

896 SD POSIX2_SW_DEV
897 The system supports the Software Development Utilities option.

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

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

ar
make
nm
strip

UP POSIX2_UPE

The system supports the User Portability Utilities option.

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

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

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

XSI XOPEN_UNIX

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

UU XOPEN_UUCP

The system supports the UUCP Utilities option.

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

uucp
uustat
uux

2.2 Application Conformance

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

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

2.2.1 Strictly Conforming POSIX Application

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

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

4. Shall not use facilities designated as *obsolescent*
5. Is required to tolerate and permitted to adapt to the presence or absence of optional facilities whose availability is indicated by Section 2.1.3 (on page 17)
6. For the C programming language, shall not produce any output dependent on any behavior described in the ISO C standard as *unspecified*, *undefined*, or *implementation-defined*, unless the System Interfaces volume of POSIX.1-2024 specifies the behavior
7. For the C programming language, shall not exceed any minimum implementation limit defined in the ISO C standard, unless the System Interfaces volume of POSIX.1-2024 specifies a higher minimum implementation limit
8. For the C programming language, shall define _POSIX_C_SOURCE to be 202405L before any header is included

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

2.2.2 Conforming POSIX Application

2.2.2.1 ISO/IEC Conforming POSIX Application

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

2.2.2.2 <National Body> Conforming POSIX Application

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

2.2.3 Conforming POSIX Application Using Extensions

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

2.2.4 Strictly Conforming XSI Application

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

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

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

2.2.5 Conforming XSI Application Using Extensions

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

2.3 Language-Dependent Services for the C Programming Language

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

2.4 Other Language-Related Specifications

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

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

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

Chapter 3
Definitions

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

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

3.1 Abortive Release

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

3.2 Absolute Pathname

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

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

3.3 Access Mode

A particular form of access permitted to a file.

3.4 Additional File Access Control Mechanism

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

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

3.5 Address Space

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

3.6 Advisory Information

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

3.7 Affirmative Response

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

Note: The *LC_MESSAGES* category is defined in detail in Section 7.3.6 (on page 159).

3.8 Alert

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

3.9 Alert Character (<alert>)

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

3.10 Alias Name

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

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

Note: The Portable Character Set is defined in detail in Section 6.1 (on page 117).

3.11 Alignment

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

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

3.12 Alternate File Access Control Mechanism

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

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

3.13 Alternate Signal Stack

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

3.14 Ancillary Data

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

3.15 Angle Brackets

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

3.16 Anonymous Memory Object

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

3.17 Apostrophe Character (<apostrophe>)

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

3.18 Application

A computer program that performs some desired function.

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

3.19 Application Address

Endpoint address of a specific application.

3.20 Application Program Interface (API)

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

3.21 Appropriate Privileges

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

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

3.22 Argument

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

Note: The Utility Argument Syntax is defined in detail in Section 12.1 (on page 213) and XCU Section 2.9.1.4 (on page 2502).

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

3.23 Arm (a Timer)

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

3.24 Asterisk Character (<asterisk>)

The character `'*'`.

3.25 Async-Cancel-Safe Function

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

3.26 Asynchronous Events

Events that occur independently of the execution of the application.

3.27 Asynchronous Input and Output

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

3.28 Async-Signal-Safe Function

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

Note: Async-signal-safety is defined in detail in XSH Section 2.4.3 (on page 516).

3.29 Asynchronously-Generated Signal

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

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

3.30 Asynchronous I/O Completion

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

3.31 Asynchronous I/O Operation

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

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

3.32 Atomic Operation

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

3.33 Authentication

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

3.34 Authorization

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

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

3.35 Background Job

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

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

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

Note: See also Section 3.158 (on page 54), Section 3.180 (on page 57), Section 3.181 (on page 57), and Section 3.362 (on page 84).

3.36 Background Process

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

3.37 Background Process Group

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

Note: See also Section 3.35.

3.38 Backquote Character

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

3.39 Backslash Character (<backslash>)

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

3.40 Backspace Character (<backspace>)

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

Note: Special Characters are defined in detail in Section 11.1.9 (on page 203).

3.41 Barrier

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

3.42 Basename

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

3.43 Basic Regular Expression (BRE)

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

Note: Basic Regular Expressions are described in detail in Section 9.3 (on page 181).

3.44 Bind

The process of assigning a network address to an endpoint.

3.45 Blank Character (<blank>)

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

3.46 Blank Line

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

3.47 Blocked Process (or Thread)

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

processor) to be satisfied before it can continue execution.

3.48 Blocking

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

3.49 Block-Mode Terminal

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

Note: Output Devices and Terminal Types are defined in detail in Section 10.2 (on page 197).

3.50 Block Special File

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

3.51 Braces

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

3.52 Brackets

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

3.53 Broadcast

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

3.54 Built-In Utility (or Built-In)

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

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

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

3.55 Byte

An individually addressable unit of data storage that is exactly an octet, used to store a character or a portion of a character; see also Section 3.58. A byte is composed of a contiguous sequence of 8 bits. The least significant bit is called the ``low-order'' bit; the most significant is called the ``high-order'' bit.

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

3.56 Byte Input/Output Functions

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

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

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

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

3.58 Character

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

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

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

3.59 Character Array

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

3.60 Character Class

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

Note: The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 131).

3.61 Character Set

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

3.62 Character Special File

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

Note: The General Terminal Interface is defined in detail in Chapter 11 (on page 199).

3.63 Character String

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

3.64 Child Process

A new process created (by *fork*(), *posix_spawn*(), or *posix_spawnp*()) by a given process. A child process remains the child of the creating process as long as both processes continue to exist.

Note: The *fork*(), *posix_spawn*(), and *posix_spawnp*() functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

3.65 Circumflex Character (<circumflex>)

The character `'^'`.

3.66 Clock

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

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

3.67 Clock Jump

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

3.68 Clock Tick

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

3.69 Code Block

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

3.70 Coded Character Set

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

3.71 Codeset

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

Note: Character Encoding is defined in detail in Section 6.2 (on page 120).

3.72 Collating Element

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

3.73 Collation

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

3.74 Collation Sequence

The relative order of collating elements as determined by the setting of the *LC_COLLATE* category in the current locale. The collation sequence is used for sorting and is determined from the collating weights assigned to each collating element. In the absence of weights, the collation sequence is the order in which collating elements are specified between **order_start** and **order_end** keywords in the *LC_COLLATE* category.

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

Note: {COLL_WEIGHTS_MAX} is defined in detail in **<limits.h>**.

3.75 Column Position

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

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

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

3.76 Command

A directive to the shell to perform a particular task.

Note: Shell Commands are defined in detail in XCU Section 2.9 (on page 2499).

3.77 Command Language Interpreter

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

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

The *system*( ), *popen*( ), and *exec* functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

3.78 Composite Graphic Symbol

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

3.79 Condition Variable

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

There are two types of condition variable: those of type **pthread_cond_t** which are initialized using *pthread_cond_init*( ) and those of type **cnd_t** which are initialized using *cnd_init*( ). If an application attempts to use the two types interchangeably (that is, pass a condition variable of type **pthread_cond_t** to a function that takes a **cnd_t**, or vice versa), the behavior is undefined.

Note: The *pthread_cond_init*( ) and *cnd_init*( ) functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

3.80 Connected Socket

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

3.81 Connection

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

3.82 Connection Mode

The transfer of data in the context of a connection; see also Section 3.83.

3.83 Connectionless Mode

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

3.84 Control Character

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

3.85 Control Operator

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

`&` `&&` `(` `)` `;` `;;` `;&` `newline` `|` `||`

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

Note: Token Recognition is defined in detail in XCU Section 2.3 (on page 2475).

3.86 Controlling Process

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

3.87 Controlling Terminal

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

Note: The General Terminal Interface is defined in detail in Chapter 11 (on page 199).

3.88 Conversion Descriptor

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

3.89 Core Image

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

3.90 CPU Time (Execution Time)

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

3.91 CPU-Time Clock

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

3.92 CPU-Time Timer

A timer attached to a CPU-time clock.

3.93 Current Job

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

3.94 Current Working Directory

See *Working Directory* in Section 3.421 (on page 93).

3.95 Cursor Position

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

3.96 Datagram

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

3.97 Data Race

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

3.98 Data Segment

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

3.99 Decimal-Point Character

See *Radix Character* in Section 3.294 (on page 74).

3.100 Declaration Utility

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

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

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

3.101 Device

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

3.102 Device ID

A non-negative integer used to identify a device.

3.103 Directory

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

3.104 Directory Entry (or Hard Link)

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

3.105 Directory Stream

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

3.106 Disarm (a Timer)

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

3.107 Display

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

3.108 Display Line

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

Note: This may also be written as ``line on the display''.

3.109 Dollar-Sign Character (<dollar-sign>)

The character `'$'`.

3.110 Dot

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

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

Pathname Resolution is defined in detail in Section 4.16 (on page 105).

3.111 Dot-Dot

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

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

3.112 Dot-Po File

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

3.113 Double-Quote Character

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

Note: The ``double'' adjective in this term refers to the two strokes in the character glyph. POSIX.1-2024 never uses the term ``double-quote'' to refer to two apostrophes or quotation-marks.

3.114 Downshifting

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

3.115 Driver

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

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

3.116 Effective Group ID

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

3.117 Effective User ID

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

3.118 Eight-Bit Transparency

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

3.119 Empty Directory

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

3.120 Empty Line

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

3.121 Empty String (or Null String)

A string whose first byte is a null byte.

3.122 Empty Wide-Character String

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

3.123 Encoding Rule

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

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

3.124 Entire Regular Expression

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

Note: Regular Expressions are defined in detail in Chapter 9 (on page 179).

3.125 Epoch

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

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

3.126 Equivalence Class

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

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

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

3.127 Era

A locale-specific method for counting and displaying years.

Note: The *LC_TIME* category is defined in detail in Section 7.3.5 (on page 152).

3.128 Event Management

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

3.129 Executable File

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

3.130 Execute

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

Note: Command Search and Execution is defined in detail in XCU Section 2.9.1.4 (on page 2502).

3.131 Execution Time

See *CPU Time* in Section 3.90 (on page 44).

3.132 Execution Time Monitoring

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

3.133 Expand

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

Note: Word Expansions are defined in detail in XCU Section 2.6 (on page 2483).

3.134 Extended Regular Expression (ERE)

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

Note: Extended Regular Expressions are described in detail in Section 9.4 (on page 187).

3.135 Extended Security Controls

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

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

File Access Permissions are defined in detail in Section 4.7 (on page 97).

3.136 Feature Test Macro

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

Note: See also XSH Section 2.2 (on page 496).

3.137 Field

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

Note: Parameter Expansion is defined in detail in XCU Section 2.6.2 (on page 2485).

Arithmetic Expansion is defined in detail in XCU Section 2.6.4 (on page 2490).

Command Substitution is defined in detail in XCU Section 2.6.3 (on page 2489).

Field Splitting is defined in detail in XCU Section 2.6.5 (on page 2491).

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

3.138 FIFO Special File (or FIFO)

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

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

3.139 File

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

3.140 File Description

See *Open File Description* in Section 3.241 (on page 66).

3.141 File Descriptor

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

Note: {OPEN_MAX} is defined in detail in **<limits.h>**.

3.142 File Group Class

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

3.143 File Lock

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

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

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

3.144 File Mode

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

Note: File mode bits and file types are defined in detail in **<sys/stat.h>**.

3.145 File Mode Bits

A file's file permission bits, set-user-ID-on-execution bit (S_ISUID), set-group-ID-on-execution bit (S_ISGID), and, on directories, the restricted deletion flag bit (S_ISVTX).

Note: File Mode Bits are defined in detail in **<sys/stat.h>**.

3.146 Filename

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

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

3.147 Filename String

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

3.148 File Offset

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

3.149 File Other Class

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

3.150 File Owner Class

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

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

3.151 File Permission Bits

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

Note: File modes are defined in detail in **<sys/stat.h>**.

File Access Permissions are defined in detail in Section 4.7 (on page 97).

3.152 File Serial Number

A per-file system unique identifier for a file.

3.153 File System

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

3.154 File Type

See *File* in Section 3.139 (on page 51).

3.155 Filter

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

3.156 First Open (of a File)

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

3.157 Flow Control

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

3.158 Foreground Job

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

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

Note: See also Section 3.35 (on page 36), Section 3.180 (on page 57), and Section 3.362 (on page 84).

3.159 Foreground Process

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

3.160 Foreground Process Group

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

Note: The General Terminal Interface is defined in detail in Chapter 11.

Note: See also Section 3.158.

3.161 Foreground Process Group ID

The process group ID of the foreground process group.

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

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

3.163 Graphic Character

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

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

3.164 Group Database

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

- Group name
- Numerical group ID
- List of users allowed in the group

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

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

3.165 Group ID

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

3.166 Group Name

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

3.167 Hard Limit

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

3.168 Hard Link

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

3.169 Hole

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

3.170 Home Directory

The directory specified by the *HOME* environment variable.

3.171 Host Byte Order

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

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

3.172 Incomplete Line

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

3.173 Inf

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

3.174 Interactive Device

A terminal device.

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

3.175 Interactive Shell

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

3.176 Internationalization

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

3.177 Interprocess Communication

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

3.178 Intrinsic Utility

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

Note: Intrinsic Utilities are defined in detail in XCU Section 1.7 (on page 2470).

3.179 Invoke

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

Note: Command Search and Execution is defined in detail in XCU Section 2.9.1.4 (on page 2502).

3.180 Job

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

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

Note: Job control in the shell is defined in detail in XCU Section 2.11 (on page 2518).

Note: See also Section 3.35 (on page 36), Section 3.158 (on page 54), and Section 3.362 (on page 84).

3.181 Job Control

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

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

Note: Job control in the shell is defined in detail in XCU Section 2.11 (on page 2518).

3.182 Job ID

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

Table 3-1 Job ID Formats

Job ID	Meaning
%%	**Current job.**
%+	Current job.
%−	Previous job.
%*n*	Job number *n*.
%*string*	Job whose command begins with *string*.
%?*string*	Job whose command contains *string*.

3.183 Joinable Thread

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

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

3.184 Last Close (of a File)

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

3.185 Line

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

3.186 Linger

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

3.187 Link

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

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

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

3.188 Link Count

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

3.189 Live Process

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

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

3.190 Live Thread

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

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

3.191 Local Customs

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

3.192 Local Interprocess Communication (Local IPC)

The transfer of data between processes in the same system.

3.193 Locale

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

Note: Locales are defined in detail in Chapter 7 (on page 127).

3.194 Localization

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

3.195 Lock-Free Operation

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

3.196 Login

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

3.197 Login Name

A user name that is associated with a login.

3.198 Map

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

3.199 Matched

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

Note: Regular Expressions are defined in detail in Chapter 9 (on page 179).

3.200 Memory Mapped Files

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

3.201 Memory Object

One of:

- A file (see Section 3.139, on page 51)
- An anonymous memory object (see Section 3.16, on page 33)
- A shared memory object (see Section 3.332, on page 80)
- A typed memory object (see Section 3.401, on page 90)

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

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

3.202 Memory-Resident

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

3.203 Message

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

3.204 Message Catalog

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

3.205 Message Catalog Descriptor

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

3.206 Message Queue

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

3.207 Messages Object

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

See also Section 3.386 (on page 88).

3.208 Mode

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

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

3.209 Monotonic Clock

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

3.210 Mount Point

Either the system root directory or a directory for which the *st_dev* field of structure **stat** differs from that of its parent directory.

Note: The **stat** structure is defined in detail in **<sys/stat.h>**.

3.211 Multi-Character Collating Element

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

3.212 Multi-Threaded Library

A library containing object files that were produced by compiling with *c17* using the flags output by *getconf* POSIX_V8_THREADS_CFLAGS, or by compiling using a non-standard utility with equivalent flags, and which makes use of interfaces that are only made available by *c17* when the **–l pthread** option is used or makes use of SIGEV_THREAD notifications.

3.213 Multi-Threaded Process

A process that contains more than one thread.

3.214 Multi-Threaded Program

A program whose executable file was produced by compiling with *c17* using the flags output by *getconf* POSIX_V8_THREADS_CFLAGS, and linking with *c17* using the flags output by *getconf* POSIX_V8_THREADS_LDFLAGS and the **–l pthread** option, or by compiling and linking using a non-standard utility with equivalent flags. Execution of a multi-threaded program initially creates a single-threaded process; the process can create additional threads using *pthread_create*( ), *thrd_create*( ), or SIGEV_THREAD notifications.

3.215 Mutex

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

There are two types of mutex: those of type **pthread_mutex_t** which are initialized using *pthread_mutex_init*( ) and those of type **mtx_t** which are initialized using *mtx_init*( ). If an application attempts to use the two types interchangeably (that is, pass a mutex of type **pthread_mutex_t** to a function that takes a **mtx_t**, or vice versa), the behavior is undefined.

Note: The *pthread_mutex_init*( ) and *mtx_init*( ) functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

3.216 Name

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

Note: The Portable Character Set is defined in detail in Section 6.1 (on page 117).

3.217 NaN (Not a Number)

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

3.218 Native Language

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

3.219 Negative

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

3.220 Negative Response

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

Note: The *LC_MESSAGES* category is defined in detail in Section 7.3.6 (on page 159).

3.221 Network

A collection of interconnected hosts.

Note: The term ``network'' in POSIX.1-2024 is used to refer to the network of hosts. The term ``batch system'' is used to refer to the network of batch servers.

3.222 Network Address

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

3.223 Network Byte Order

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

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

3.224 Newline Character (<newline>)

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

3.225 Nice Value

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

3.226 Non-Blocking

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

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

3.227 Non-Spacing Characters

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

3.228 NUL

A character with all bits set to zero.

3.229 Null Byte

A byte with all bits set to zero.

3.230 Null Pointer

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

3.231 Null String

See *Empty String* in Section 3.121 (on page 48).

3.232 Null Terminator

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

3.233 Null Wide-Character Code

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

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

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

3.235 Object File

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

3.236 Octet

Unit of data representation that consists of eight contiguous bits.

3.237 OFD-Owned File Lock

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

3.238 Offset Maximum

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

3.239 Opaque Address

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

3.240 Open File

A file that is currently associated with a file descriptor.

3.241 Open File Description

A record of how a process or group of processes is accessing a file. Each file descriptor refers to exactly one open file description, but an open file description can be referred to by more than one file descriptor. The file offset, file status, and file access modes are attributes of an open file description.

3.242 Operand

An argument to a command that is generally used as an object supplying information to a utility necessary to complete its processing. Operands generally follow the options in a command line.

Note: Utility Argument Syntax is defined in detail in Section 12.1 (on page 213).

3.243 Operator

In the shell command language, either a control operator or a redirection operator.

3.244 Option

An argument to a command that is generally used to specify changes in the utility's default behavior.

Note: Utility Argument Syntax is defined in detail in Section 12.1 (on page 213).

3.245 Option-Argument

A parameter that follows certain options. In some cases an option-argument immediately follows the option character within the same argument string as the option; otherwise the option-argument is the next argument string.

Note: Utility Argument Syntax is defined in detail in Section 12.1 (on page 213).

3.246 Orientation

A stream has one of three orientations: unoriented, byte-oriented, or wide-oriented.

Note: For further information, see XSH Section 2.5.2 (on page 524).

3.247 Orphaned Process Group

A process group in which the parent of every member is either itself a member of the group or is not a member of the group's session.

3.248 Page

The granularity of process memory mapping or locking.

Physical memory and memory objects can be mapped into the address space of a process on page boundaries and in integral multiples of pages. Process address space can be locked into memory (made memory-resident) on page boundaries and in integral multiples of pages.

3.249 Page Size

The size, in bytes, of the system unit of memory allocation, protection, and mapping. On systems that have segment rather than page-based memory architectures, the term ``page'' means a segment.

3.250 Parameter

In the shell command language, an entity that stores values. There are three types of parameters: variables (named parameters), positional parameters, and special parameters. Parameter expansion is accomplished by introducing a parameter with the `'$'` character.

Note: See also XCU Section 2.5 (on page 2478).

In the C language, an object declared as part of a function declaration or definition that acquires a value on entry to the function, or an identifier following the macro name in a function-like macro definition.

3.251 Parent Directory

When discussing a given directory, the directory that both contains a directory entry for the given directory and is represented by the pathname dot-dot in the given directory.

When discussing other types of files, a directory containing a directory entry for the file under discussion.

This concept does not apply to dot and dot-dot.

3.252 Parent Process

The process which created (or inherited) the process under discussion.

3.253 Parent Process ID

An attribute of a new process identifying the parent of the process. The parent process ID of a process is the process ID of its creator, for the lifetime of the creator. After the creator's lifetime has ended, the parent process ID is the process ID of an implementation-defined system process.

3.254 Pathname

A string that is used to identify a file. In the context of POSIX.1-2024, a pathname may be limited to {PATH_MAX} bytes, including the terminating null byte. It has optional beginning <slash> characters, followed by zero or more filenames separated by <slash> characters. A pathname can optionally contain one or more trailing <slash> characters. Multiple successive <slash> characters are considered to be the same as one <slash>, except it is implementation-defined whether the case of exactly two leading <slash> characters is treated specially.

Note: If a pathname consists of only bytes corresponding to characters from the portable filename character set (see Section 3.265, on page 70), <slash> characters, and a single terminating <NUL> character, the pathname will be usable as a character string in all supported locales; otherwise, the pathname might only be a string (rather than a character string). Additionally, since the single-byte encoding of the <slash> character is required to be the same across all locales and to not occur within a multi-byte character, references to a <slash> character within a pathname are well-defined even when the pathname is not a character string. However, this property does not necessarily hold for the remaining characters within the portable filename character set.

Pathname Resolution is defined in detail in Section 4.16 (on page 105).

3.255 Pathname Component

See *Filename* in Section 3.146 (on page 52).

3.256 Path Prefix

The part of a pathname up to, but not including, the last component and any trailing <slash> characters, unless the pathname consists entirely of <slash> characters, in which case the path prefix is `'/'` for a pathname containing either a single <slash> or three or more <slash> characters, and `'//'` for the pathname **//**. The path prefix of a pathname containing no <slash> characters is empty, but is treated as referring to the current working directory.

Note: The term is used both in the sense of identifying part of a pathname that forms the prefix and of joining a non-empty path prefix to a filename to form a pathname. In the latter case, the path prefix need not have a trailing <slash> (in which case the joining is done with a <slash> character).

3.257 Pattern

A sequence of characters used either with regular expression notation or with shell pattern matching notation.

Note: Regular Expressions are defined in detail in Chapter 9 (on page 179).

Shell pattern matching notation is defined in detail in Section 2.14 (on page 2523).

The syntaxes of the two types of patterns are similar, but not identical; POSIX.1-2024 always indicates the type of pattern being referred to in the immediate context of the use of the term.

3.258 Period Character (<period>)

The character `'.'`. The term ``period'' is contrasted with dot (see also Section 3.110, on page 47), which is used to describe a specific directory entry.

3.259 Permissions

Attributes of an object that determine the privilege necessary to access or manipulate the object.

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

3.260 Persistence

A mode for semaphores, shared memory, and message queues requiring that the object and its state (including data, if any) are preserved after the object is no longer referenced by any process.

Persistence of an object does not imply that the state of the object is maintained across a system crash or a system reboot.

3.261 Pipe

An object identical to a FIFO which has no links in the file hierarchy.

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

3.262 Polling

A scheduling scheme whereby the local process periodically checks until the pre-specified events (for example, read, write) have occurred.

3.263 Portable Character Set

The collection of characters that are required to be present in all locales supported by conforming systems.

Note: The Portable Character Set is defined in detail in Section 6.1 (on page 117).

This term is contrasted against the smaller portable filename character set; see also Section 3.265.

3.264 Portable Filename

A filename consisting only of characters from the portable filename character set.

Note: Applications should avoid using filenames that have the <hyphen-minus> character as the first character since this may cause problems when filenames are passed as command line arguments.

3.265 Portable Filename Character Set

The set of characters from which portable filenames are constructed.

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -
```

The last three characters are the <period>, <underscore>, and <hyphen-minus> characters, respectively. See also Section 3.254 (on page 68).

3.266 Portable Messages Object Source File (or Dot-Po File)

A text file containing messages and directives. A portable messages object source file can be compiled into a messages object by the *msgfmt* utility.

Note: By convention, portable messages object source files have filenames ending with the **.po** suffix. Utility descriptions in this standard frequently use dot-po file as a shorthand for portable messages object source file (even though the **.po** suffix need not be included in the filename). Template portable messages object source files can be created from C-language source files by the *xgettext* utility.

3.267 Positional Parameter

In the shell command language, a parameter denoted by a decimal representation of a positive integer.

Note: For further information, see XCU Section 2.5.1 (on page 2479).

3.268 Positive

When describing a value (not a sign), greater than zero. Note that in the common phrase ``positive zero'' (which is not used in this standard, although the representation +0.0 is) it describes a sign, and therefore positive zero (+0.0) is not a positive value.

3.269 Preallocation

The reservation of resources in a system for a particular use.

Preallocation does not imply that the resources are immediately allocated to that use, but merely indicates that they are guaranteed to be available in bounded time when needed.

3.270 Preempted Process (or Thread)

A running thread whose execution is suspended due to another thread becoming runnable at a higher priority.

3.271 Previous Job

In the context of job control, the job used as the default for the *fg* or *bg* utilities if the current job exits. There is at most one previous job; see also Section 3.182 (on page 57).

3.272 Printable Character

One of the characters included in the **print** character classification of the *LC_CTYPE* category in the current locale.

Note: The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 131).

3.273 Printable File

A text file consisting only of the characters included in the **print** and **space** character classifications of the *LC_CTYPE* category and the <backspace>, all in the current locale.

Note: The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 131).

3.274 Priority

A non-negative integer associated with processes or threads whose value is constrained to a range defined by the applicable scheduling policy. Numerically higher values represent higher priorities.

3.275 Priority Inversion

A condition in which a thread that is not voluntarily suspended (waiting for an event or time delay) is not running while a lower priority thread is running. Such blocking of the higher priority thread is often caused by contention for a shared resource.

3.276 Priority Scheduling

A performance and determinism improvement facility to allow applications to determine the order in which threads that are ready to run are granted access to processor resources.

3.277 Priority-Based Scheduling

Scheduling in which the selection of a running thread is determined by the priorities of the runnable processes or threads.

3.278 Privilege

See *Appropriate Privileges* in Section 3.21 (on page 34).

3.279 Process

A live process (see Section 3.189, on page 59) or a zombie process (see Section 3.426, on page 94). The lifetime of a process is described in Section 3.285 (on page 73).

3.280 Process Group

A collection of processes that permits the signaling of related processes. Each process in the system is a member of a process group that is identified by a process group ID. A newly created process joins the process group of its creator.

3.281 Process Group ID

The unique positive integer identifier representing a process group during its lifetime.

Note: See also *Process Group ID Reuse* defined in Section 4.17 (on page 106).

3.282 Process Group Leader

A process whose process ID is the same as its process group ID.

3.283 Process Group Lifetime

The period of time that begins when a process group is created and ends when the last remaining process in the group leaves the group, due either to the end of the lifetime of the last process or to the last remaining process calling the *setsid*( ) or *setpgid*( ) functions.

Note: The *setsid*( ) and *setpgid*( ) functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

3.284 Process ID

The unique positive integer identifier representing a process during its lifetime.

Note: See also *Process ID Reuse* defined in Section 4.17 (on page 106).

3.285 Process Lifetime

The period of time that begins when a process is created and ends when its process ID is returned to the system.

See also Section 3.189 (on page 59), Section 3.287, and Section 3.426 (on page 94).

Note: Process creation is defined in detail in the descriptions of the *fork*( ), *posix_spawn*( ), and *posix_spawnp*( ) functions in the System Interfaces volume of POSIX.1-2024.

3.286 Process Memory Locking

A performance improvement facility to bind application programs into the high-performance random access memory of a computer system. This avoids potential latencies introduced by the operating system in storing parts of a program that were not recently referenced on secondary memory devices.

3.287 Process Termination

There are two kinds of process termination:

1. Normal termination occurs by a return from *main*( ), when requested with the *exit*( ), *_exit*( ), or *_Exit*( ) functions; or when the last thread in the process terminates by returning from its start function, by calling the *pthread_exit*( ) or *thrd_exit*( ) function, or through cancellation.
2. Abnormal termination occurs when requested by the *abort*( ) function or when some signals are received.

Note: The consequences of process termination can be found in the description of the *_Exit*( ) function in the System Interfaces volume of POSIX.1-2024. The *_exit*( ), *_Exit*( ), *abort*( ), and *exit*( ) functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

3.288 Process Virtual Time

The measurement of time in units elapsed by the system clock while a process is executing.

3.289 Process-Owned File Lock

A record lock owned by a process. Process-owned file locks are obtained through the use of *fcntl*( ) with F_SETLK or F_SETLKW, or the use of *lockf*( ). Process-owned file locks are not inherited by child processes, but are preserved across the *exec* family of functions. A process-owned file lock is released when the process exits, or when any file descriptor in the process referring to the same file is closed (even if via a different open file description). These locks are shared among all open file descriptions referring to the same file in the process, making the use of process-owned file locks unsuitable for use for coordination of record access among multiple threads in a process.

3.290 Process-To-Process Communication

The transfer of data between processes.

3.291 Program

A prepared sequence of instructions to the system to accomplish a defined task. The term ``program'' in POSIX.1-2024 encompasses applications written in the Shell Command Language, complex utility input languages (for example, *awk*, *lex*, *sed*, and so on), and high-level languages.

3.292 Protocol

A set of semantic and syntactic rules for exchanging information.

3.293 Pseudo-Terminal

A facility that provides an interface that is identical to the terminal subsystem, except where noted otherwise in POSIX.1-2024. A pseudo-terminal is composed of two devices: the ``manager device'' and a ``subsidiary device''. The subsidiary device provides processes with an interface that is identical to the terminal interface, although there need not be hardware behind that interface. Anything written on the manager device is presented to the subsidiary as an input and anything written on the subsidiary device is presented as an input on the manager side.

3.294 Radix Character (or Decimal-Point Character)

The character that separates the integer part of a number from the fractional part.

3.295 Read-Only File System

A file system that has implementation-defined characteristics restricting modifications.

Note: File Times Update is described in detail in Section 4.12 (on page 98).

3.296 Read-Write Lock

Multiple readers, single writer (read-write) locks allow many threads to have simultaneous read-only access to data while allowing only one thread to have write access at any given time. They are typically used to protect data that is read-only more frequently than it is changed.

Read-write locks can be used to synchronize threads in the current process and other processes if they are allocated in memory that is writable and shared among the cooperating processes and have been initialized for this behavior.

3.297 Real Group ID

The attribute of a process that, at the time of process creation, identifies the group of the user who created the process; see also Section 3.165 (on page 55).

3.298 Real Time

Time measured as total units elapsed by the system clock without regard to which thread is executing.

3.299 Realtime Signal Extension

A determinism improvement facility to enable asynchronous signal notifications to an application to be queued without impacting compatibility with the existing signal functions.

3.300 Real User ID

The attribute of a process that, at the time of process creation, identifies the user who created the process; see also Section 3.408 (on page 91).

3.301 Record

A collection of related data units or words which is treated as a unit.

3.302 Record Lock

A file lock held on a record within a file. A record lock can be used to lock a whole file by specifying a special record with starting offset zero and length zero. (This special record extends to any future end-of-file, not just the current end-of-file.) This includes an OFD-owned file lock (see Section 3.237, on page 66) or a process-owned file lock (see Section 3.289, on page 74). It is unspecified whether an implementation will detect and prevent deadlocks caused by two competing lock owners holding separate locks where each tries to obtain a lock that is blocked by the other's lock.

3.303 Redirection

In the shell command language, a method of associating files with the input or output of commands.

Note: For further information, see XCU Section 2.7 (on page 2493).

3.304 Redirection Operator

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

`<` `>` `>|` `<<` `>>` `<&` `>&` `<<-` `<>`

3.305 Referenced Shared Memory Object

A shared memory object that is open or has one or more mappings defined on it.

3.306 Refresh

Make the information on the user's terminal screen up-to-date.

3.307 Regular Built-In Utility (or Regular Built-In)

See *Built-In Utility* in Section 3.54 (on page 39).

3.308 Regular Expression

A pattern that selects specific strings from a set of character strings.

Note: Regular Expressions are described in detail in Chapter 9 (on page 179).

3.309 Region

In the context of the address space of a process, a sequence of addresses.

In the context of a file, a sequence of offsets.

3.310 Regular File

A file that is a randomly accessible sequence of bytes, with no further structure imposed by the system.

3.311 Relative Pathname

A pathname not beginning with a <slash> character.

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

3.312 Relocatable File

A file holding code or data suitable for linking with other object files to create an executable or a shared object file.

3.313 Relocation

The process of connecting symbolic references with symbolic definitions. For example, when a program calls a function, the associated call instruction transfers control to the proper destination address at execution.

3.314 (Time) Resolution

The minimum time interval that a clock can measure or whose passage a timer can detect.

3.315 Robust Mutex

A mutex with the *robust* attribute set.

Note: The *robust* attribute is defined in detail by the *pthread_mutexattr_getrobust*( ) function.

3.316 Root Directory

A directory, associated with a process, that is used in pathname resolution for pathnames that begin with a <slash> character.

3.317 Runnable Process (or Thread)

A thread that is capable of being a running thread, but for which no processor is available.

3.318 Running Process (or Thread)

A thread currently executing on a processor. On multi-processor systems there may be more than one such thread in a system at a time.

3.319 Saved Resource Limits

An attribute of a process that provides some flexibility in the handling of unrepresentable resource limits, as described in the *exec* family of functions and *setrlimit*( ).

Note: The *exec* and *setrlimit*( ) functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

3.320 Saved Set-Group-ID

An attribute of a process that allows some flexibility in the assignment of the effective group ID attribute, as described in the *exec* family of functions and *setgid*( ).

Note: The *exec* and *setgid*( ) functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

3.321 Saved Set-User-ID

An attribute of a process that allows some flexibility in the assignment of the effective user ID attribute, as described in the *exec* family of functions and *setuid*( ).

Note: The *exec* and *setuid*( ) functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

3.322 Scheduling

The application of a policy to select a runnable process or thread to become a running process or thread, or to alter one or more of the thread lists.

3.323 Scheduling Allocation Domain

The set of processors on which an individual thread can be scheduled at any given time.

3.324 Scheduling Contention Scope

A property of a thread that defines the set of threads against which that thread competes for resources.

For example, in a scheduling decision, threads sharing scheduling contention scope compete for processor resources. In POSIX.1-2024, a thread has scheduling contention scope of either PTHREAD_SCOPE_SYSTEM or PTHREAD_SCOPE_PROCESS.

3.325 Scheduling Policy

A set of rules that is used to determine the order of execution of processes or threads to achieve some goal.

Note: Scheduling Policy is defined in detail in Section 4.18 (on page 107).

3.326 Screen

A rectangular region of columns and lines on a terminal display. A screen may be a portion of a physical display device or may occupy the entire physical area of the display device.

3.327 Scroll

To move the representation of data vertically or horizontally relative to the terminal screen. There are two types of scrolling:

1. The cursor moves with the data.
2. The cursor remains stationary while the data moves.

3.328 Semaphore

A minimum synchronization primitive to serve as a basis for more complex synchronization mechanisms to be defined by the application program.

Note: Semaphores are defined in detail in Section 4.20 (on page 108).

3.329 Session

A collection of process groups established for job control purposes. Each process group is a member of a session. A process is considered to be a member of the session of which its process group is a member. A newly created process joins the session of its creator. A process can alter its session membership; see *setsid*( ). There can be multiple process groups in the same session.

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

3.330 Session Leader

A process that has created a session.

Note: For further information, see the *setsid*( ) function defined in the System Interfaces volume of POSIX.1-2024.

3.331 Session Lifetime

The period between when a session is created and the end of the lifetime of all the process groups that remain as members of the session.

3.332 Shared Memory Object

An object that represents memory that can be mapped concurrently into the address space of more than one process.

3.333 Shell

A program that interprets sequences of text input as commands. It may operate on an input stream or it may interactively prompt and read commands from a terminal.

3.334 Shell, the

The Shell Command Language Interpreter; a specific instance of a shell.

Note: For further information, see the *sh* utility defined in the Shell and Utilities volume of POSIX.1-2024.

3.335 Shell Script

A file containing shell commands. If the file is made executable, it can be executed by specifying its name as a simple command. Execution of a shell script causes a shell to execute the commands within the script. Alternatively, a shell can be requested to execute the commands in a shell script by specifying the name of the shell script as the operand to the *sh* utility.

Note: Simple Commands are defined in detail in XCU Section 2.9.1 (on page 2500).

The *sh* utility is defined in detail in the Shell and Utilities volume of POSIX.1-2024.

3.336 Signal

A mechanism by which a process or thread may be notified of, or affected by, an event occurring in the system. Examples of such events include hardware exceptions and specific actions by processes. The term signal is also used to refer to the event itself.

3.337 Signal Stack

Memory established for a thread, in which signal handlers catching signals sent to that thread are executed.

3.338 Single-Quote Character

The character designated by `'\''` in the C language, also known as <apostrophe>.

3.339 Single-Threaded Process

A process that contains a single thread.

3.340 Single-Threaded Program

A program whose executable file was produced by compiling with *c17* without using the flags output by *getconf* POSIX_V8_THREADS_CFLAGS and linking with *c17* using neither the flags output by *getconf* POSIX_V8_THREADS_LDFLAGS nor the **–l pthread** option, or by compiling and linking using a non-standard utility with equivalent flags. Execution of a single-threaded program creates a single-threaded process; if the process attempts to create additional threads using *pthread_create*( ), *thrd_create*( ), or SIGEV_THREAD notifications, the behavior is undefined. If the process uses *dlopen*( ) to load a multi-threaded library, the behavior is undefined.

3.341 Slash Character (<slash>)

The character `'/'`, also known as solidus.

3.342 Socket

A file of a particular type that is used as a communications endpoint for process-to-process communication as described in the System Interfaces volume of POSIX.1-2024.

3.343 Socket Address

An address associated with a socket or remote endpoint, including an address family identifier and addressing information specific to that address family. The address may include multiple parts, such as a network address associated with a host system and an identifier for a specific endpoint.

3.344 Soft Limit

A resource limitation established for each process that the process may set to any value less than or equal to the hard limit.

3.345 Source Code

When dealing with the Shell Command Language, input to the command language interpreter. The term ``shell script'' is synonymous with this meaning.

When dealing with an ISO/IEC-conforming programming language, source code is input to a compiler conforming to that ISO/IEC standard.

Source code also refers to the input statements prepared for the following standard utilities: *awk*, *bc*, *ed*, *ex*, *lex*, *localedef*, *make*, *sed*, and *yacc*.

Source code can also refer to a collection of sources meeting any or all of these meanings.

Note: The *awk*, *bc*, *ed*, *ex*, *lex*, *localedef*, *make*, *sed*, and *yacc* utilities are defined in detail in the Shell and Utilities volume of POSIX.1-2024.

3.346 Space Character (<space>)

The character defined in the portable character set as <space>. The <space> character is a member of the **space** character class of the current locale, but represents the single character, and not all of the possible members of the class; see also Section 3.413 (on page 92).

3.347 Sparse File

A file that contains more holes than just the virtual hole at the end of the file.

3.348 Spawn

A process creation primitive useful for systems that have difficulty with *fork*( ) and as an efficient replacement for *fork*( )/*exec*.

3.349 Special Built-In Utility (or Special Built-In)

See *Built-In Utility* in Section 3.54 (on page 39).

3.350 Special Parameter

In the shell command language, a parameter named by a single character from the following list:

```
*   @   #   ?   !   -   $   0
```

Note: For further information, see XCU Section 2.5.2 (on page 2479).

3.351 Spin Lock

A synchronization object used to allow multiple threads to serialize their access to shared data.

3.352 Sporadic Server

A scheduling policy for threads and processes that reserves a certain amount of execution capacity for processing aperiodic events at a given priority level.

3.353 Standard Error

In the context of file descriptors (see Section 3.141, on page 51), file descriptor number 2.

In the context of standard I/O streams (see XSH Section 2.5, on page 521), an output stream usually intended to be used for diagnostic messages, and accessed using the global variable *stderr*.

Note: The file descriptor underlying *stderr* is initially 2, but it can be changed by *freopen*( ) to 0 or 1 (and implementations may have extensions that allow it to be changed to other numbers). Therefore, writing to the standard error stream does not always produce output on the standard error file descriptor.

3.354 Standard Input

In the context of file descriptors (see Section 3.141, on page 51), file descriptor number 0.

In the context of standard I/O streams (see XSH Section 2.5, on page 521), an input stream usually intended to be used for primary data input, and accessed using the global variable *stdin*.

Note: The file descriptor underlying *stdin* is initially 0; this cannot change through the use of interfaces defined in this standard, but implementations may have extensions that allow it to be changed. Therefore, in conforming applications using extensions, reading from the standard input stream does not always obtain input from the standard input file descriptor.

3.355 Standard Output

In the context of file descriptors (see Section 3.141, on page 51), file descriptor number 1.

In the context of standard I/O streams (see XSH Section 2.5, on page 521), an output stream usually intended to be used for primary data output, and accessed using the global variable *stdout*.

Note: The file descriptor underlying *stdout* is initially 1, but it can be changed by *freopen*( ) to 0 (and implementations may have extensions that allow it to be changed to other numbers). Therefore, writing to the standard output stream does not always produce output on the standard output file descriptor.

3.356 Standard Utilities

The utilities described in the Shell and Utilities volume of POSIX.1-2024.

3.357 Stream

Appearing in lowercase, a stream is an ordered sequence of bytes, as described by the ISO C standard.

In the shell command language, each stream is associated with a file descriptor. These can be opened using redirection operators.

Note: Redirection is defined in detail in XCU Section 2.7 (on page 2493).

In the C language, each stream is accessed via a file access object and is either a stream associated with a file descriptor or a memory stream. A file access object associated with a file descriptor can be created by the *fdopen*( ), *fopen*( ), or *popen*( ) functions. A file access object for a memory stream can be created by the *fmemopen*( ) or *open_memstream*( ) functions. A stream provides the additional services of user-selectable buffering and formatted input and output.

Note: For further information, see XSH Section 2.5 (on page 521).

The *fdopen*( ), *fmemopen*( ), *fopen*( ), *open_memstream*( ), and *popen*( ) functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

3.358 String

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

3.359 Subshell

A shell execution environment, distinguished from the main or current shell execution environment.

Note: For further information, see XCU Section 2.13 (on page 2522).

3.360 Successfully Transferred

For a write operation to a regular file, when the system ensures that all data written is readable on any subsequent open of the file (even one that follows a system or power failure) in the absence of a failure of the physical storage medium.

For a read operation, when an image of the data on the physical storage medium is available to the requesting process.

3.361 Supplementary Group ID

An attribute of a process used in determining file access permissions. A process has up to {NGROUPS_MAX} supplementary group IDs in addition to the effective group ID. The supplementary group IDs of a process are set to the supplementary group IDs of the parent process when the process is created.

3.362 Suspended Job

In the context of the System Interfaces volume of POSIX.1-2024, a job that has received a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal that caused the process group to stop.

In the context of the shell, a job, other than a non-job-control background job, that became suspended when a process returned a wait status to the shell indicating that the process was stopped by a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal.

A suspended job is a job-control background job, but a job-control background job is not necessarily a suspended job. A non-job-control background job is never a suspended job, even if it includes processes that have been stopped by a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal.

Note: See also Section 3.35 (on page 36), Section 3.180 (on page 57), and Section 3.158 (on page 54).

3.363 Symbolic Constant

An object-like macro defined with a constant value.

Unless stated otherwise, the following shall apply to every symbolic constant:

- It expands to a compile-time constant expression with an integer type.
- It may be defined as another type of constant—e.g., an enumeration constant—as well as being a macro.
- It need not be usable in **#if** preprocessing directives.

3.364 Symbolic Link

A type of file with the property that when the file is encountered during pathname resolution, a string stored by the file is used to modify the pathname resolution. The stored string has a length of {SYMLINK_MAX} bytes or fewer.

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

3.365 Synchronization Operation

An operation that synchronizes memory. See Section 4.15 (on page 100).

3.366 Synchronized Input and Output

A determinism and robustness improvement mechanism to enhance the data input and output mechanisms, so that an application can be assured that the data being manipulated is physically present on secondary mass storage devices.

3.367 Synchronized I/O Completion

The state of an I/O operation that has either been successfully transferred or diagnosed as unsuccessful.

3.368 Synchronized I/O Data Integrity Completion

For read, when the operation has been completed or diagnosed if unsuccessful. The read is complete only when an image of the data has been successfully transferred to the requesting process. If there were any pending write requests affecting the data to be read at the time that the synchronized read operation was requested, these write requests are successfully transferred prior to reading the data.

For write, when the operation has been completed or diagnosed if unsuccessful. The write is complete only when the data specified in the write request is successfully transferred and all file system information required to retrieve the data is successfully transferred.

For the purpose of this definition, an operation that reads or searches a directory is considered to be a read operation, an operation that modifies a directory is considered to be a write operation, and a directory's entries are considered to be the data read or written.

This standard provides no way to synchronize the contents or attributes of a symbolic link.

File attributes that are not necessary for data retrieval (access time, modification time, status change time) need not be successfully transferred prior to returning to the calling process.

3.369 Synchronized I/O File Integrity Completion

Identical to a synchronized I/O data integrity completion with the addition that all file attributes relative to the I/O operation (including access time, modification time, status change time) are successfully transferred prior to returning to the calling process.

3.370 Synchronized I/O Operation

An I/O operation performed on a file that provides the application assurance of the integrity of its data and files.

3.371 Synchronous I/O Operation

An I/O operation that causes the thread requesting the I/O to be blocked from further use of the processor until that I/O operation completes.

Note: A synchronous I/O operation does not imply synchronized I/O data integrity completion or synchronized I/O file integrity completion.

3.372 Synchronously-Generated Signal

A signal that is attributable to a specific thread.

For example, a thread executing an illegal instruction or touching invalid memory causes a synchronously-generated signal. Being synchronous is a property of how the signal was generated and not a property of the signal number.

3.373 System

An implementation of POSIX.1-2024.

3.374 System Boot

An unspecified sequence of events that may result in the loss of transitory data; that is, data that is not saved in permanent storage. For example, message queues, shared memory, semaphores, and processes.

3.375 System Clock

A clock with at least one second resolution that contains seconds since the Epoch.

3.376 System Console

A device that receives messages sent by the *syslog*() function, and the *fmtmsg*() function when the MM_CONSOLE flag is set.

Note: The *syslog*() and *fmtmsg*() functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

3.377 System Crash

An interval initiated by an unspecified circumstance that causes all processes (possibly other than special system processes) to be terminated in an undefined manner, after which any

changes to the state and contents of files created or written to by an application prior to the interval are undefined, except as required elsewhere in POSIX.1-2024.

3.378 System Databases

An implementation provides two system databases: the ``group database'' (see also Section 3.164, on page 55) and the ``user database'' (see also Section 3.407, on page 91).

3.379 System Documentation

All documentation provided with an implementation except for the conformance document. Electronically distributed documents for an implementation are considered part of the system documentation.

3.380 System Process

An object other than a process executing an application, that is provided by the system and has a process ID.

3.381 System Reboot

See *System Boot* defined in Section 3.374 (on page 86).

3.382 System-Wide

Pertaining to events occurring in all processes existing in an implementation at a given point in time.

3.383 Tab Character (<tab>)

A character that in the output stream indicates that printing or displaying should start at the next horizontal tabulation position on the current line. It is the character designated by `'\t'` in the C language. If the current position is at or past the last defined horizontal tabulation position, the behavior is unspecified. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the tabulation.

3.384 Terminal (or Terminal Device)

A character special file that obeys the specifications of the general terminal interface.

Note: The General Terminal Interface is defined in detail in Chapter 11 (on page 199).

3.385 Text Column

A roughly rectangular block of characters capable of being laid out side-by-side next to other text columns on an output page or terminal screen. The widths of text columns are measured in

column positions.

3.386 Text Domain

A named collection of messages objects (one messages object per supported language) for internationalization and localization purposes. A text domain is often named after the application or library that provides the collection, but may have a more general name if it is intended to be shared by multiple applications or libraries.

Note: The use of text domains is defined in detail in the descriptions of the *bindtextdomain*( ) and *gettext* family of functions in the System Interfaces volume of POSIX.1-2024.

3.387 Text File

A file that contains characters organized into zero or more lines. The lines do not contain NUL characters and none can exceed {LINE_MAX} bytes in length, including the <newline> character. Although POSIX.1-2024 does not distinguish between text files and binary files (see the ISO C standard), many utilities only produce predictable or meaningful output when operating on text files. The standard utilities that have such restrictions always specify ``text files'' in their STDIN or INPUT FILES sections.

3.388 Thread

A live thread (see Section 3.190, on page 59) or a zombie thread (see Section 3.427, on page 94). The lifetime of a thread is described in Section 3.390 (on page 89).

3.389 Thread ID

A value that uniquely identifies each thread in a process during the thread's lifetime. The value shall be unique across all threads in a process, regardless of whether the thread is:

- The initial thread
- A thread created using *pthread_create*( )
- A thread created using *thrd_create*( )
- A thread created via a SIGEV_THREAD notification

Note: Since *pthread_create*( ) returns an ID of type **pthread_t** and *thrd_create*( ) returns an ID of type **thrd_t**, this uniqueness requirement necessitates that these two types are defined as the same underlying type because calls to *pthread_self*( ) and *thrd_current*( ) from the initial thread need to return the same thread ID. The *pthread_create*( ), *pthread_self*( ), *thrd_create*( ), and *thrd_current*( ) functions and SIGEV_THREAD notifications are defined in detail in the System Interfaces volume of POSIX.1-2024.

3.390 Thread Lifetime

The period of time that begins when a thread is created and ends when its thread ID is returned to the process.

See also *Live Thread* in Section 3.190 (on page 59), *Thread Termination* in Section 3.392, and *Zombie Thread* in Section 3.427 (on page 94).

Note: Thread creation is defined in detail in the descriptions of the *pthread_create*( ) and *thrd_create*( ) functions in the System Interfaces volume of POSIX.1-2024.

3.391 Thread List

An ordered set of runnable threads that all have the same ordinal value for their priority.

The ordering of threads on the list is determined by a scheduling policy or policies. The set of thread lists includes all runnable threads in the system.

3.392 Thread Termination

Thread termination occurs when a thread executes *pthread_exit*( ) or *thrd_exit*( ), when it returns from the *start_routine* function passed to *pthread_create*( ) or from the *func* function passed to *thrd_create*( ), or when it acts on a cancellation request initiated by *pthread_cancel*( ).

Note: The *pthread_cancel*( ), *pthread_create*( ), *pthread_exit*( ), *thrd_create*( ), and *thrd_exit*( ) functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

3.393 Thread-Safe

A thread-safe function shall avoid data races with other calls to the same function, and with calls to any other thread-safe functions, by multiple threads. Each function defined in the System Interfaces volume of POSIX.1-2024 is thread-safe unless explicitly stated otherwise. Examples are any ``pure'' function, a function which holds a mutex locked while it is accessing static storage, or objects shared among threads.

A function that is not required to be thread-safe need not avoid data races with other calls to the same function, nor with calls to any other function (including thread-safe functions), by multiple threads, unless explicitly stated otherwise.

3.394 Thread-Specific Data Key

A process global handle which is used for naming thread-specific data. There are two types of key: those of type **pthread_key_t** which are created using *pthread_key_create*( ) and those of type **tss_t** which are created using *tss_create*( ). If an application attempts to use the two types of key interchangeably (that is, pass a key of type **pthread_key_t** to a function that takes a **tss_t**, or vice versa), the behavior is undefined.

Although the same key value can be used by different threads, the values bound to the key by *pthread_setspecific*( ) for keys of type **pthread_key_t**, and by *tss_set*( ) for keys of type **tss_t**, are maintained on a per-thread basis and persist for the life of the calling thread.

Note: The *pthread_getspecific*( ), *pthread_setspecific*( ), *tss_create*( ), and *tss_set*( ) functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

3.395 Tilde Character (<tilde>)

The character '~'.

3.396 Timeouts

A method of limiting the length of time an interface will block; see also Section 3.47 (on page 37).

3.397 Timer

A mechanism that can notify a thread when the time as measured by a particular clock has reached or passed a specified value, or when a specified amount of time has passed.

3.398 Timer Overrun

A condition that occurs each time a timer, for which there is already an expiration signal queued to the process, expires.

3.399 Token

In the shell command language, a sequence of characters that the shell considers as a single unit when reading input. A token is either an operator or a word.

Note: The rules for reading input are defined in detail in XCU Section 2.3 (on page 2475).

3.400 Typed Memory Name Space

A system-wide name space that contains the names of the typed memory objects present in the system. It is configurable for a given implementation.

3.401 Typed Memory Object

A combination of a typed memory pool and a typed memory port. The entire contents of the pool are accessible from the port. The typed memory object is identified through a name that belongs to the typed memory name space.

3.402 Typed Memory Pool

An extent of memory with the same operational characteristics. Typed memory pools may be contained within each other.

3.403 Typed Memory Port

A hardware access path to one or more typed memory pools.

3.404 Unbind

Remove the association between a network address and an endpoint.

3.405 Unit Data

See *Datagram* in Section 3.96 (on page 45).

3.406 Upshifting

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

3.407 User Database

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

- User name
- Numerical user ID
- Initial numerical group ID
- Initial working directory
- Initial user program

The initial numerical group ID is used by the *newgrp* utility. Any other circumstances under which the initial values are operative are implementation-defined.

If the initial user program field is null, an implementation-defined program is used.

If the initial working directory field is null, the interpretation of that field is implementation-defined.

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

3.408 User ID

A non-negative integer that is used to identify a system user. When the identity of a user is associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or a saved set-user-ID. The value (**uid_t**)–1 shall not be a valid user ID, but does have a defined use in some interfaces defined in this standard.

3.409 User Name

A string that is used to identify a user; see also Section 3.407. To be portable across systems conforming to POSIX.1-2024, the value is composed of characters from the portable filename

character set. The <hyphen-minus> character should not be used as the first character of a portable user name.

3.410 Utility

A program, excluding special built-in utilities provided as part of the Shell Command Language, that can be called by name from a shell to perform a specific task, or related set of tasks.

Note: For further information on special built-in utilities, see XCU Section 2.15 (on page 2526).

3.411 Variable

In the shell command language, a named parameter.

Note: For further information, see XCU Section 2.5 (on page 2478).

3.412 Vertical-Tab Character (<vertical-tab>)

A character that in the output stream indicates that printing should start at the next vertical tabulation position. It is the character designated by `'\v'` in the C language. If the current position is at or past the last defined vertical tabulation position, the behavior is unspecified. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the tabulation.

3.413 White Space

A sequence of one or more characters that belong to the **space** character class as defined via the *LC_CTYPE* category in the current locale or a specified locale.

In the POSIX locale, white space consists of one or more <blank> (<space> and <tab> characters), <newline>, <carriage-return>, <form-feed>, and <vertical-tab> characters.

3.414 White-Space Byte

A single-byte white-space character; that is, a character for which the *isspace*() or *isspace_l*() function returns a non-zero value.

3.415 White-Space Character

A character that belongs to the **space** character class as defined via the *LC_CTYPE* category in the current locale or a specified locale.

3.416 White-Space Wide Character

A wide-character code that belongs to the **space** character class as defined via the *LC_CTYPE* category in the current locale or a specified locale.

3.417 Wide-Character Code (C Language)

An integer value corresponding to a single graphic symbol or control code.

Note: C Language Wide-Character Codes are defined in detail in Section 6.3 (on page 120).

3.418 Wide-Character Input/Output Functions

The functions that perform wide-oriented input from streams or wide-oriented output to streams: *fgetwc*( ), *fgetws*( ), *fputwc*( ), *fputws*( ), *fwprintf*( ), *fwscanf*( ), *getwc*( ), *getwchar*( ), *putwc*( ), *putwchar*( ), *ungetwc*( ), *vfwprintf*( ), *vfwscanf*( ), *vwprintf*( ), *vwscanf*( ), *wprintf*( ), and *wscanf*( ).

Note: These functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

3.419 Wide-Character String

A contiguous sequence of wide-character codes terminated by and including the first null wide-character code.

3.420 Word

In the shell command language, a token other than an operator. In some cases a word is also a portion of a word token: in the various forms of parameter expansion, such as ${*name–word*}, and variable assignment, such as *name=word*, the word is the portion of the token depicted by *word*. The concept of a word is no longer applicable following word expansions—only fields remain.

Note: For further information, see XCU Section 2.6.2 (on page 2485) and Section 2.6 (on page 2483).

3.421 Working Directory (or Current Working Directory)

A directory, associated with a process, that is used in pathname resolution for pathnames that do not begin with a <slash> character.

3.422 Worldwide Portability Interface

Functions for handling characters in a codeset-independent manner.

3.423 Write

To output characters to a file, such as standard output or standard error. Unless otherwise stated, standard output is the default output destination for all uses of the term ``write''; see the distinction between display and write in Section 3.107 (on page 46).

3.424 XSI

The X/Open System Interfaces (XSI) option is the core application programming interface for C

and *sh* programming for systems conforming to the Single UNIX Specification. This is a superset of the mandatory requirements for conformance to POSIX.1-2024.

3.425 XSI-Conformant

A system which allows an application to be built using a set of services that are consistent across all systems that conform to POSIX.1-2024 and that support the XSI option.

Note: See also Chapter 2 (on page 15).

3.426 Zombie Process

The remains of a live process (see Section 3.189, on page 59) after it terminates (see Section 3.287, on page 73) and before its status information (see XSH Section 2.12, on page 563) is consumed by its parent process.

3.427 Zombie Thread

The remains of a joinable live thread (see Section 3.183 (on page 58) and Section 3.190, on page 59) after it terminates (see Section 3.392, on page 89) and before it has been joined with *pthread_join*( ) or *thrd_join*( ) or detached with *pthread_detach*( ) or *thrd_detach*( ).

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

3.428 ±0

The algebraic sign provides additional information about any variable that has the value zero when the representation allows the sign to be determined.

Chapter 4
General Concepts

For the purposes of POSIX.1-2024, the general concepts given in Chapter 4 apply.

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

4.1 Case Insensitive Comparisons

When a standard utility or function that uses regular expressions or pattern matching specifies that matching shall be case insensitive, then if a string would match the regular expression or pattern when doing a case-sensitive match, the same string with any of its characters replaced with their case counterparts, as defined by the **toupper** and **tolower** character mappings (see Section 7.3.1, on page 131), shall also match when doing a case-insensitive match.

This definition of case-insensitive processing is intended to allow matching of multi-character collating elements as well as characters, as each character in the string is matched using both its cases. For example, in a locale with a `"Ch"` multi-character collating element (see Section 7.3.2, on page 139), the bracket expression `"[[.Ch.]]"` (see Section 9.3.5 (on page 182) item 4) matches the strings `"ch"`, `"Ch"`, `"cH"`, and `"CH"` when matching without regard to case.

4.2 Concurrent Execution

Functions that suspend the execution of the calling thread shall not cause the execution of other threads to be indefinitely suspended.

4.3 Default Initialization

Default initialization causes an object to be initialized according to these rules:

- If it has pointer type, it is initialized to a null pointer.
- If it has arithmetic type, it is initialized to (positive or unsigned) zero.
- If it is an aggregate, every member is initialized (recursively) according to these rules.
- If it is a union, the first named member is initialized (recursively) according to these rules.

For an object of aggregate type with an explicit initializer, the initialization shall occur in initializer list order, each initializer provided for a particular subobject overriding any previously listed initializer for the same subobject; all subobjects that are not initialized explicitly shall be initialized implicitly according to the rules for default initialization.

Objects with static storage duration but no explicit initializer shall be initialized implicitly according to the rules for default initialization.

An explicit initializer of `{ 0 }` works to perform explicit default initialization for any object of scalar or aggregate type, and for any storage duration.

Notes:

1. The ISO C standard does not require a compiler to set any field alignment padding bits in a structure or array definition to a particular value. Because of this, a structure initialized using `{ 0 }` might not *memcmp*() as equal to the same structure initialized using *memset*() to zero. For consistent results, portable applications comparing structures should test each field individually.
2. If an implementation treats the all-zero bit pattern of a floating-point object as equivalent to positive 0, then *memset*() to zero and *calloc*() have the same effects as default initialization for all named members of a structure. Implementations that define __STDC_IEC_559__ guarantee that the all-zero bit pattern of a floating-point object represents 0.0.

4.4 Directory Operations

All file system operations that read or search a directory or that modify the contents of a directory (for example creating, unlinking, or renaming a file) shall operate atomically. That is, each operation shall either have its entire effect and succeed, or shall not affect the file system and shall fail. Furthermore, these operations shall be serializable; that is, the state of the file system and of the results of each operation shall always be values that would be obtained if the operations were executed one after the other.

4.5 Directory Protection

If a directory is writable and the mode bit S_ISVTX is set on the directory, a process may remove or rename files within that directory only if one or more of the following is true:

- The effective user ID of the process is the same as that of the owner ID of the file.
- The effective user ID of the process is the same as that of the owner ID of the directory.
- The process has appropriate privileges.
- Optionally, the file is writable by the process. Whether or not files that are writable by the process can be removed or renamed is implementation-defined.

If the S_ISVTX bit is set on a non-directory file, the behavior is unspecified.

4.6 Extended Security Controls

An implementation may provide implementation-defined extended security controls (see Section 3.135, on page 50). These permit an implementation to provide security mechanisms to implement different security policies than those described in POSIX.1-2024. These mechanisms shall not alter or override the defined semantics of any of the interfaces in POSIX.1-2024.

4.7 File Access Permissions

The standard file access control mechanism uses the file permission bits, as described below.

Implementations may provide *additional* or *alternate* file access control mechanisms, or both. An additional access control mechanism shall only further restrict the access permissions defined by the file permission bits. An alternate file access control mechanism shall:

- Specify file permission bits for the file owner class, file group class, and file other class of that file, corresponding to the access permissions.
- Be enabled only by explicit user action, on a per-file basis by the file owner or a user with appropriate privileges.
- Be disabled for a file after the file permission bits are changed for that file with *chmod*(). The disabling of the alternate mechanism need not disable any additional mechanisms supported by an implementation.

Whenever a process requests file access permission for read, write, or execute/search, if no additional mechanism denies access, access shall be determined as follows:

- If a process has appropriate privileges:
  - — If read, write, or directory search permission is requested, access shall be granted.
  - — If execute permission is requested, access shall be granted if execute permission is granted to at least one user by the file permission bits or by an alternate access control mechanism; otherwise, access shall be denied.
- Otherwise:
  - — The file permission bits of a file contain read, write, and execute/search permissions for the file owner class, file group class, and file other class.
  - — Access shall be granted if an alternate access control mechanism is not enabled and the requested access permission bit is set for the class (file owner class, file group class, or file other class) to which the process belongs, or if an alternate access control mechanism is enabled and it allows the requested access; otherwise, access shall be denied.

4.8 File Hierarchy

Files in the system are organized in a hierarchical structure in which all of the non-terminal nodes are directories and all of the terminal nodes are any other type of file. Since multiple directory entries may refer to the same file, the hierarchy is properly described as a ``directed graph''.

4.9 Filenames

Uppercase and lowercase letters shall retain their unique identities between conforming implementations.

4.10 Filename Portability

For a filename to be portable across implementations conforming to POSIX.1-2024, it shall consist only of the portable filename character set as defined in Section 3.265 (on page 70).

Note: Applications should avoid using filenames that have the <hyphen-minus> character as the first character since this may cause problems when filenames are passed as command line arguments.

4.11 File System Cache

If the file system is accessed via a memory cache, file-related requirements stated in the rest of this standard shall apply to the cache, except where explicitly stated otherwise: this includes directory atomicity and serializability requirements (see Section 4.4), file times update requirements (see Section 4.12), and read-write serializability requirements (see *write*()). Cache entries shall be transferred to the underlying storage as the result of successful calls to *fdatasync*(), *fsync*(), or *aio_fsync*(), and may be transferred to storage automatically at other times. Such transfers shall be atomic, with minimum units being directory entries (for directory contents), aligned data blocks of the fundamental file system block size (for regular-file contents; see **<sys/statvfs.h>**), and all attributes of a single file (for file attributes).

Note: If the system crashes before the cache is fully transferred, later operations' effects may be present in storage with earlier effects missing.

Note: Operations that create or modify multiple directory entries, aligned data blocks, or file attributes (e.g., *mkdir*(), *rename*(), *write*() with large buffer size, *open*() with O_CREAT) may have only part of their effects transferred to storage, and after a crash these operations may appear to have been only partly done, with the parts not necessarily done in any order. For example, only the second half of a *write*() may be transferred; or `rename("a","b")` may result in **b** being created without **a** being removed.

Note: Although conforming file systems are required to perform all caching as described above, some file systems may support non-conforming configurations (for example via mount options) for which this is not the case. Applications that are used on non-conforming file systems cannot rely on files being synchronized properly.

4.12 File Times Update

Many operations have requirements to update file timestamps. These requirements do not apply to streams that have no underlying file description (for example, memory streams created by *open_memstream*() have no underlying file description).

Each file has three distinct associated timestamps: the time of last data access, the time of last data modification, and the time the file status last changed. These values are returned in the file characteristics structure **struct stat**, as described in **<sys/stat.h>** (on page 414).

Each function or utility in POSIX.1-2024 that reads or writes data (even if the data does not change) or performs an operation to change file status (even if the file status does not change) indicates which of the appropriate timestamps shall be marked for update. If an implementation of such a function or utility marks for update one of these timestamps in a place or time not specified by POSIX.1-2024, this shall be documented, except that any changes caused by pathname resolution need not be documented. For the other functions or utilities in POSIX.1-2024 (those that are not explicitly required to read or write file data or change file status, but that in some implementations happen to do so), the effect is unspecified.

An implementation may update timestamps that are marked for update immediately, or it may update such timestamps periodically. At the point in time when an update occurs, any marked timestamps shall be set to the current time and the update marks shall be cleared. All timestamps that are marked for update shall be updated when the file ceases to be open by any process or before a *fstat*( ), *fstatat*( ), *fsync*( ), *futimens*( ), *lstat*( ), *stat*( ), *utimensat*( ), or *utimes*( ) is successfully performed on the file. Other times at which updates are done are unspecified. Marks for update, and updates themselves, shall not be done for files on read-only file systems; see Section 3.295 (on page 75).

The resolution of timestamps of files in a file system is implementation-defined, but shall be no coarser than one-second resolution. The three timestamps shall always have values that are supported by the file system. Whenever any of a file's timestamps are to be set to a value V according to the rules of the preceding paragraphs of this section, the implementation shall immediately set the timestamp to the greatest value supported by the file system that is not greater than V.

4.13 Host and Network Byte Orders

When data is transmitted over the network, it is sent as a sequence of octets (8-bit unsigned values). If an entity (such as an address or a port number) can be larger than 8 bits, it needs to be stored in several octets. The convention is that all such values are stored with 8 bits in each octet, and with the first (lowest-addressed) octet holding the most-significant bits. This is called ``network byte order''.

Network byte order may not be convenient for processing actual values. For this, it is more sensible for values to be stored as ordinary integers. This is known as ``host byte order''. In host byte order:

- The most significant bit might not be stored in the first byte in address order.
- Bits might not be allocated to bytes in any obvious order at all.

8-bit values stored in **uint8_t** objects do not require conversion to or from host byte order, as they have the same representation. 16 and 32-bit values can be converted using the *htonl*( ), *htons*( ), *ntohl*( ), and *ntohs*( ) functions. When reading data that is to be converted to host byte order, it should either be received directly into a **uint16_t** or **uint32_t** object or should be copied from an array of bytes using *memcpy*( ) or similar. Passing the data through other types could cause the byte order to be changed. Similar considerations apply when sending data.

4.14 Measurement of Execution Time

The mechanism used to measure execution time shall be implementation-defined. The implementation shall also define to whom the CPU time that is consumed by interrupt handlers and system services on behalf of the operating system will be charged. See Section 3.90 (on page 44).

4.15 Memory Ordering and Synchronization

4.15.1 Memory Ordering

4.15.1.1 Data Races

The value of an object visible to a thread *T* at a particular point is the initial value of the object, a value stored in the object by *T*, or a value stored in the object by another thread, according to the rules below.

Two expression evaluations *conflict* if one of them modifies a memory location and the other one reads or modifies the same memory location.

This standard defines a number of atomic operations (see **<stdatomic.h>**) and operations on mutexes (see **<threads.h>**) that are specially identified as synchronization operations. These operations play a special role in making assignments in one thread visible to another. A synchronization operation on one or more memory locations is either an *acquire operation*, a *release operation*, both an acquire and release operation, or a *consume operation*. A synchronization operation without an associated memory location is a *fence* and can be either an acquire fence, a release fence, or both an acquire and release fence. In addition, there are *relaxed atomic operations*, which are not synchronization operations, and atomic *read-modify-write operations*, which have special characteristics.

Note: For example, a call that acquires a mutex will perform an acquire operation on the locations composing the mutex. Correspondingly, a call that releases the same mutex will perform a release operation on those same locations. Informally, performing a release operation on *A* forces prior side effects on other memory locations to become visible to other threads that later perform an acquire or consume operation on *A*. Relaxed atomic operations are not included as synchronization operations although, like synchronization operations, they cannot contribute to data races.

All modifications to a particular atomic object *M* occur in some particular total order, called the modification order of *M*. If *A* and *B* are modifications of an atomic object *M*, and *A* happens before *B*, then *A* shall precede *B* in the modification order of *M*, which is defined below.

Note: This states that the modification orders must respect the ``happens before'' relation.

Note: There is a separate order for each atomic object. There is no requirement that these can be combined into a single total order for all objects. In general this will be impossible since different threads may observe modifications to different variables in inconsistent orders.

A *release sequence* headed by a release operation *A* on an atomic object *M* is a maximal contiguous sub-sequence of side effects in the modification order of *M*, where the first operation is *A* and every subsequent operation either is performed by the same thread that performed the release or is an atomic read-modify-write operation.

Certain system interfaces *synchronize with* other system interfaces performed by another thread. In particular, an atomic operation *A* that performs a release operation on an object *M* shall synchronize with an atomic operation *B* that performs an acquire operation on *M* and reads a value written by any side effect in the release sequence headed by *A*.

Note: Except in the specified cases, reading a later value does not necessarily ensure visibility as described below. Such a requirement would sometimes interfere with efficient implementation.

Note: The specifications of the synchronization operations define when one reads the value written by another. For atomic variables, the definition is clear. All operations on a given mutex occur in a single total order. Each mutex acquisition ``reads the value written'' by the last mutex release.

An evaluation *A carries a dependency* to an evaluation *B* if:

- the value of *A* is used as an operand of *B*, unless:
  - *B* is an invocation of the *kill_dependency*( ) macro,
  - *A* is the left operand of a `&&` or `||` operator,
  - *A* is the left operand of a `?:` operator, or
  - *A* is the left operand of a `,` (comma) operator; or
- *A* writes a scalar object or bit-field *M*, *B* reads from *M* the value written by *A*, and *A* is sequenced before *B*, or
- for some evaluation *X*, *A* carries a dependency to *X* and *X* carries a dependency to *B*.

An evaluation *A* is *dependency-ordered before* an evaluation *B* if:

- *A* performs a release operation on an atomic object *M*, and, in another thread, *B* performs a consume operation on *M* and reads a value written by any side effect in the release sequence headed by *A*, or
- for some evaluation *X*, *A* is dependency-ordered before *X* and *X* carries a dependency to *B*.

An evaluation *A inter-thread happens before* an evaluation *B* if *A* synchronizes with *B*, *A* is dependency-ordered before *B*, or, for some evaluation *X*:

- *A* synchronizes with *X* and *X* is sequenced before *B*,
- *A* is sequenced before *X* and *X* inter-thread happens before *B*, or
- *A* inter-thread happens before *X* and *X* inter-thread happens before *B*.

Note: The ``inter-thread happens before'' relation describes arbitrary concatenations of ``sequenced before'', ``synchronizes with'', and ``dependency-ordered before'' relationships, with two exceptions. The first exception is that a concatenation is not permitted to end with ``dependency-ordered before'' followed by ``sequenced before''. The reason for this limitation is that a consume operation participating in a ``dependency-ordered before'' relationship provides ordering only with respect to operations to which this consume operation actually carries a dependency. The reason that this limitation applies only to the end of such a concatenation is that any subsequent release operation will provide the required ordering for a prior consume operation. The second exception is that a concatenation is not permitted to consist entirely of ``sequenced before''. The reasons for this limitation are (1) to permit ``inter-thread happens before'' to be transitively closed and (2) the ``happens before'' relation, defined below, provides for relationships consisting entirely of ``sequenced before''.

An evaluation *A happens before* an evaluation *B* if *A* is sequenced before *B* or *A* inter-thread happens before *B*. The implementation shall ensure that a cycle in the ``happens before'' relation never occurs.

Note: This cycle would otherwise be possible only through the use of consume operations.

A *visible side effect A* on an object *M* with respect to a value computation *B* of *M* satisfies the conditions:

- *A* happens before *B*, and
- there is no other side effect *X* to *M* such that *A* happens before *X* and *X* happens before *B*.

The value of a non-atomic scalar object *M*, as determined by evaluation *B*, shall be the value

stored by the visible side effect *A*.

Note: If there is ambiguity about which side effect to a non-atomic object is visible, then there is a data race and the behavior is undefined.

Note: This states that operations on ordinary variables are not visibly reordered. This is not actually detectable without data races, but it is necessary to ensure that data races, as defined here, and with suitable restrictions on the use of atomics, correspond to data races in a simple interleaved (sequentially consistent) execution.

The value of an atomic object *M*, as determined by evaluation *B*, shall be the value stored by some side effect *A* that modifies *M*, where *B* does not happen before *A*.

Note: The set of side effects from which a given evaluation might take its value is also restricted by the rest of the rules described here, and in particular, by the coherence requirements below.

If an operation *A* that modifies an atomic object *M* happens before an operation *B* that modifies *M*, then *A* shall be earlier than *B* in the modification order of *M*. (This is known as ``write-write coherence''.)

If a value computation *A* of an atomic object *M* happens before a value computation *B* of *M*, and *A* takes its value from a side effect *X* on *M*, then the value computed by *B* shall either be the value stored by *X* or the value stored by a side effect *Y* on *M*, where *Y* follows *X* in the modification order of *M*. (This is known as ``read-read coherence''.)

If a value computation *A* of an atomic object *M* happens before an operation *B* on *M*, then *A* shall take its value from a side effect *X* on *M*, where *X* precedes *B* in the modification order of *M*. (This is known as ``read-write coherence''.)

If a side effect *X* on an atomic object *M* happens before a value computation *B* of *M*, then the evaluation *B* shall take its value from *X* or from a side effect *Y* that follows *X* in the modification order of *M*. (This is known as ``write-read coherence''.)

Note: This effectively disallows implementation reordering of atomic operations to a single object, even if both operations are ``relaxed'' loads. By doing so, it effectively makes the ``cache coherence'' guarantee provided by most hardware available to POSIX atomic operations.

Note: The value observed by a load of an atomic object depends on the ``happens before'' relation, which in turn depends on the values observed by loads of atomic objects. The intended reading is that there must exist an association of atomic loads with modifications they observe that, together with suitably chosen modification orders and the ``happens before'' relation derived as described above, satisfy the resulting constraints as imposed here.

An application contains a data race if it contains two conflicting actions in different threads, at least one of which is not atomic, and neither happens before the other. Any such data race results in undefined behavior.

4.15.1.2 *Memory Order and Consistency*

The enumerated type **memory_order**, defined in **<stdatomic.h>** (if supported), specifies the detailed regular (non-atomic) memory synchronization operations as defined in Section 4.15.1.1 (on page 100) and may provide for operation ordering. Its enumeration constants specify memory order as follows:

For `memory_order_relaxed`, no operation orders memory.

For `memory_order_release`, `memory_order_acq_rel`, and `memory_order_seq_cst`, a store operation performs a release operation on the affected memory location.

For `memory_order_acquire`, `memory_order_acq_rel`, and `memory_order_seq_cst`, a load operation performs an acquire operation on the affected memory location.

For memory_order_consume, a load operation performs a consume operation on the affected memory location.

There shall be a single total order *S* on all memory_order_seq_cst operations, consistent with the ``happens before'' order and modification orders for all affected locations, such that each memory_order_seq_cst operation *B* that loads a value from an atomic object *M* observes one of the following values:

- the result of the last modification *A* of *M* that precedes *B* in *S*, if it exists, or
- if *A* exists, the result of some modification of *M* that is not memory_order_seq_cst and that does not happen before *A*, or
- if *A* does not exist, the result of some modification of *M* that is not memory_order_seq_cst.

Note: Although it is not explicitly required that *S* include lock operations, it can always be extended to an order that does include lock and unlock operations, since the ordering between those is already included in the ``happens before'' ordering.

Note: Atomic operations specifying memory_order_relaxed are relaxed only with respect to memory ordering. Implementations must still guarantee that any given atomic access to a particular atomic object be indivisible with respect to all other atomic accesses to that object.

For an atomic operation *B* that reads the value of an atomic object *M*, if there is a memory_order_seq_cst fence *X* sequenced before *B*, then *B* observes either the last memory_order_seq_cst modification of *M* preceding *X* in the total order *S* or a later modification of *M* in its modification order.

For atomic operations *A* and *B* on an atomic object *M*, where *A* modifies *M* and *B* takes its value, if there is a memory_order_seq_cst fence *X* such that *A* is sequenced before *X* and *B* follows *X* in *S*, then *B* observes either the effects of *A* or a later modification of *M* in its modification order.

For atomic modifications *A* and *B* of an atomic object *M*, *B* occurs later than *A* in the modification order of *M* if:

- there is a memory_order_seq_cst fence *X* such that *A* is sequenced before *X*, and *X* precedes *B* in *S*, or
- there is a memory_order_seq_cst fence *Y* such that *Y* is sequenced before *B*, and *A* precedes *Y* in *S*, or
- there are memory_order_seq_cst fences *X* and *Y* such that *A* is sequenced before *X*, *Y* is sequenced before *B*, and *X* precedes *Y* in *S*.

Atomic read-modify-write operations shall always read the last value (in the modification order) stored before the write associated with the read-modify-write operation.

An atomic store shall only store a value that has been computed from constants and input values by a finite sequence of evaluations, such that each evaluation observes the values of variables as computed by the last prior assignment in the sequence. The ordering of evaluations in this sequence shall be such that:

- If an evaluation *B* observes a value computed by *A* in a different thread, then *B* does not happen before *A*.
- If an evaluation *A* is included in the sequence, then all evaluations that assign to the same variable and happen before *A* are also included.

3204 **Note:** The second requirement disallows ``out-of-thin-air'', or ``speculative'' stores of atomics when
3205 relaxed atomics are used. Since unordered operations are involved, evaluations can appear in
3206 this sequence out of thread order.

3207 ### 4.15.2 Memory Synchronization

3208 In order to avoid data races, applications shall ensure that non-lock-free access to any memory
3209 location by more than one thread of control (threads or processes) is restricted such that no
3210 thread of control can read or modify a memory location while another thread of control might be
3211 modifying it. Such access can be restricted using functions that synchronize thread execution
3212 and also synchronize memory with respect to other threads. The following functions shall
3213 synchronize memory with respect to other threads on all successful calls:

3214 *cnd_broadcast*( )	*pthread_rwlock_clockrdlock*( )	*sem_timedwait*( )
3215 *cnd_signal*( )	*pthread_rwlock_clockwrlock*( )	*sem_trywait*( )
3216 *fork*( )	*pthread_rwlock_rdlock*( )	*sem_wait*( )
3217 *pthread_barrier_wait*( )	*pthread_rwlock_timedrdlock*( )	*semctl*( )
3218 *pthread_cond_broadcast*( )	*pthread_rwlock_timedwrlock*( )	*semop*( )
3219 *pthread_cond_signal*( )	*pthread_rwlock_tryrdlock*( )	*thrd_create*( )
3220 *pthread_create*( )	*pthread_rwlock_trywrlock*( )	*thrd_join*( )
3221 *pthread_join*( )	*pthread_rwlock_unlock*( )	*wait*( )
3222 *pthread_spin_lock*( )	*pthread_rwlock_wrlock*( )	*waitid*( )
3223 *pthread_spin_trylock*( )	*sem_clockwait*( )	*waitpid*( )
3224 *pthread_spin_unlock*( )	*sem_post*( )	

3225 The *pthread_once*( ) and *call_once*( ) functions shall synchronize memory for the first successful
3226 call in each thread for a given **pthread_once_t** or **once_flag** object, respectively. If the *init_routine*
3227 called by *pthread_once*( ) or *call_once*( ) is a cancellation point and is canceled, a successful call to
3228 *pthread_once*( ) for the same **pthread_once_t** object or to *call_once*( ) for the same **once_flag** object,
3229 made from a cancellation cleanup handler shall also synchronize memory.

3230 RPP|TPP The *pthread_mutex_clocklock*( ), *pthread_mutex_lock*( ), *pthread_mutex_setprioceiling*( ),
3231 *pthread_mutex_timedlock*( ), and *pthread_mutex_trylock*( ) functions shall synchronize memory on
3232 all calls that acquire the mutex, including those that return [EOWNERDEAD]. The
3233 *pthread_mutex_unlock*( ) function shall synchronize memory on all calls that release the mutex.

3234 **Note:** If the mutex type is PTHREAD_MUTEX_RECURSIVE, calls to the locking functions do not
3235 acquire the mutex if the calling thread already owns it, and calls to *pthread_mutex_unlock*( ) do
3236 not release the mutex if it has a lock count greater than one.

3237 The *pthread_cond_clockwait*( ), *pthread_cond_wait*( ), and *pthread_cond_timedwait*( ) functions shall
3238 synchronize memory on all calls that release and re-acquire the specified mutex, including calls
3239 that return [EOWNERDEAD], both when the mutex is released and when it is re-acquired.

3240 **Note:** If the mutex type is PTHREAD_MUTEX_RECURSIVE, calls to *pthread_cond_clockwait*( ),
3241 *pthread_cond_wait*( ), and *pthread_cond_timedwait*( ) do not release and re-acquire the mutex if it
3242 has a lock count greater than one.

3243 The *mtx_lock*( ), *mtx_timedlock*( ), and *mtx_trylock*( ) functions shall synchronize memory on all
3244 calls that acquire the mutex. The *mtx_unlock*( ) function shall synchronize memory on all calls
3245 that release the mutex.

3246 **Note:** If the mutex is a recursive mutex, calls to the locking functions do not acquire the mutex if the
3247 calling thread already owns it, and calls to *mtx_unlock*( ) do not release the mutex if it has a lock
3248 count greater than one.

3249 The *cnd_wait*( ) and *cnd_timedwait*( ) functions shall synchronize memory on all calls that release

and re-acquire the specified mutex, both when the mutex is released and when it is re-acquired.

Note: If the mutex is a recursive mutex, calls to *cnd_wait*( ) and *cnd_timedwait*( ) do not release and re-acquire the mutex if it has a lock count greater than one.

Unless explicitly stated otherwise, if one of the functions named in this section returns an error, it is unspecified whether the invocation causes memory to be synchronized.

Applications can allow more than one thread of control to read a memory location simultaneously.

For purposes of determining the existence of a data race, all lock and unlock operations on a particular synchronization object that synchronize memory shall behave as atomic operations, and they shall occur in some particular total order (see Section 4.15.1, on page 100).

4.16 Pathname Resolution

Pathname resolution is performed for a process to resolve a pathname to a particular directory entry for a file in the file hierarchy. There may be multiple pathnames that resolve to the same directory entry, and multiple directory entries for the same file. When a process resolves a pathname of an existing directory entry, the entire pathname shall be resolved as described below. When a process resolves a pathname of a directory entry that is to be created immediately after the pathname is resolved, pathname resolution terminates when all components of the path prefix of the last component have been resolved. It is then the responsibility of the process to create the final component.

Each filename in the pathname is located in the directory specified by its predecessor (for example, in the pathname fragment **a/b**, file **b** is located in the directory specified by **a**). Pathname resolution shall fail if this cannot be accomplished. If the pathname begins with a <slash>, the predecessor of the first filename in the pathname shall be taken to be the root directory of the process (such pathnames are referred to as ``absolute pathnames''). If the pathname does not begin with a <slash>, the predecessor of the first filename of the pathname shall be taken to be either the current working directory of the process or for certain interfaces the directory identified by a file descriptor passed to the interface (such pathnames are referred to as ``relative pathnames'').

The interpretation of a pathname component is dependent on the value of {NAME_MAX} and _POSIX_NO_TRUNC associated with the path prefix of that component. If any pathname component is longer than {NAME_MAX}, the implementation shall consider this an error.

A pathname that contains at least one non-<slash> character and that ends with one or more trailing <slash> characters shall not be resolved successfully unless the last pathname component before the trailing <slash> characters resolves (with symbolic links followed—see below) to an existing directory or a directory entry that is to be created for a directory immediately after the pathname is resolved. Interfaces using pathname resolution may specify additional constraints[7] when a pathname that does not name an existing directory contains at least one non-<slash> character and contains one or more trailing <slash> characters.

If a symbolic link is encountered during pathname resolution, the behavior shall depend on whether the pathname component is at the end of the pathname and on the function being performed. If all of the following are true, then pathname resolution is complete:

7. The only interfaces that further constrain pathnames in POSIX.1-2024 are the *rename*( ) and *renameat*( ) functions (see XSH *rename*( )) and the *mv* utility (see XCU *mv*).

3293 1. This is the last pathname component of the pathname.
3294 2. The pathname has no trailing <slash>.
3295 3. The function is required to act on the symbolic link itself, or certain arguments direct that
3296 the function act on the symbolic link itself.

3297 In all other cases, the system shall prefix the remaining pathname, if any, with the contents of the
3298 symbolic link, except that if the contents of the symbolic link is the empty string, then either
3299 pathname resolution shall fail with functions reporting an [ENOENT] error and utilities writing
3300 an equivalent diagnostic message, or the pathname of the directory containing the symbolic link
3301 shall be used in place of the contents of the symbolic link. If the contents of the symbolic link
3302 consist solely of <slash> characters, then all leading <slash> characters of the remaining
3303 pathname shall be omitted from the resulting combined pathname, leaving only the leading
3304 <slash> characters from the symbolic link contents. In the cases where prefixing occurs, if the
3305 combined length exceeds {PATH_MAX}, and the implementation considers this to be an error,
3306 pathname resolution shall fail with functions reporting an [ENAMETOOLONG] error and
3307 utilities writing an equivalent diagnostic message. Otherwise, the resolved pathname shall be
3308 the resolution of the pathname just created. If the resulting pathname does not begin with a
3309 <slash>, the predecessor of the first filename of the pathname is taken to be the directory
3310 containing the symbolic link.

3311 If the system detects a loop in the pathname resolution process, pathname resolution shall fail
3312 with functions reporting an [ELOOP] error and utilities writing an equivalent diagnostic
3313 message. The same may happen if during the resolution process more symbolic links were
3314 followed than the implementation allows. This implementation-defined limit shall not be
3315 smaller than {SYMLOOP_MAX}.

3316 The special filename dot shall refer to the directory specified by its predecessor. The special
3317 filename dot-dot shall refer to the parent directory of its predecessor directory. As a special case,
3318 in the root directory, dot-dot may refer to the root directory itself.

3319 A pathname consisting of a single <slash> shall resolve to the root directory of the process. A
3320 null pathname shall not be successfully resolved. If a pathname begins with two successive
3321 <slash> characters, the first component following the leading <slash> characters may be
3322 interpreted in an implementation-defined manner, although more than two leading <slash>
3323 characters shall be treated as a single <slash> character.

3324 Pathname resolution for a given pathname shall yield the same results when used by any
3325 interface in POSIX.1-2024 as long as there are no changes to any files evaluated during pathname
3326 resolution for the given pathname between resolutions.

3327 4.17 Process ID Reuse

3328 A process group ID shall not be reused by the system until the process group lifetime ends.

3329 A process ID shall not be reused by the system until the process lifetime ends. In addition, if
3330 there exists a process group whose process group ID is equal to that process ID, the process ID
3331 shall not be reused by the system until the process group lifetime ends. A process that is not a
3332 system process shall not have a process ID of 1.

4.18 Scheduling Policy

A scheduling policy affects process or thread ordering:

- When a process or thread is a running thread and it becomes a blocked thread
- When a process or thread is a running thread and it becomes a preempted thread
- When a process or thread is a blocked thread and it becomes a runnable thread
- When a running thread calls a function that can change the priority or scheduling policy of a process or thread
- In other scheduling policy-defined circumstances

Conforming implementations shall define the manner in which each of the scheduling policies may modify the priorities or otherwise affect the ordering of processes or threads at each of the occurrences listed above. Additionally, conforming implementations shall define in what other circumstances and in what manner each scheduling policy may modify the priorities or affect the ordering of processes or threads.

4.19 Seconds Since the Epoch

A value that approximates the number of seconds that have elapsed since the Epoch. A Coordinated Universal Time name (specified in terms of seconds (*tm_sec*), minutes (*tm_min*), hours (*tm_hour*), days since January 1 of the year (*tm_yday*), and calendar year minus 1900 (*tm_year*)) is related to a time represented as seconds since the Epoch, according to the expression below.

If the year is <1970 or the value is negative, the relationship is undefined. If the year is ≥1970 and the value is non-negative, the value is related to a Coordinated Universal Time name according to the C-language expression, where *tm_sec*, *tm_min*, *tm_hour*, *tm_yday*, and *tm_year* are all integer types:

```
tm_sec + tm_min*60 + tm_hour*3600 + tm_yday*86400 +
    (tm_year-70)*31536000 + ((tm_year-69)/4)*86400 -
    ((tm_year-1)/100)*86400 + ((tm_year+299)/400)*86400
```

The relationship between the actual date and time in Coordinated Universal Time, as determined by the International Earth Rotation Service, and the system's current value for seconds since the Epoch is unspecified.

How any changes to the value of seconds since the Epoch are made to align to a desired relationship with the current actual time is implementation-defined. As represented in seconds since the Epoch, each and every day shall be accounted for by exactly 86 400 seconds.

Note: The last three terms of the expression add in a day for each year that follows a leap year starting with the first leap year since the Epoch. The first term adds a day every 4 years starting in 1973, the second subtracts a day back out every 100 years starting in 2001, and the third adds a day back in every 400 years starting in 2001. The divisions in the formula are integer divisions; that is, the remainder is discarded leaving only the integer quotient.

4.20 Semaphore

A minimum synchronization primitive to serve as a basis for more complex synchronization mechanisms to be defined by the application program.

For the mandatory semaphores (those not associated with the X/Open System Interfaces (XSI) option), a semaphore is represented as a shareable resource that has a non-negative integer value. When the value is zero, there is a (possibly empty) set of threads awaiting the availability of the semaphore.

For the semaphores associated with the X/Open System Interfaces (XSI) option, a semaphore is an integer with minimum value 0 and an implementation-defined maximum value which shall be at least 32767. The *semget*( ) function can be called to create a set or array of semaphores. A semaphore set can contain one or more semaphores up to an implementation-defined value.

Semaphore Lock Operation

An operation that is applied to a semaphore. If, prior to the operation, the value of the semaphore is zero, the semaphore lock operation shall cause the calling thread to be blocked and added to the set of threads awaiting the semaphore; otherwise, the value shall be decremented.

Semaphore Unlock Operation

An operation that is applied to a semaphore. If, prior to the operation, there are any threads in the set of threads awaiting the semaphore, then some thread from that set shall be removed from the set and becomes unblocked; otherwise, the semaphore value shall be incremented.

4.21 Special Device Drivers

Some devices require control operations, other than the operations that are common to most devices (such as *read*( ), *write*( ), *open*( ), and *close*( )), but because the device belongs to a class that is not present in the majority of systems, standardization of a device-specific application program interface (API) for controlling it has not been practical. The driver for such a device may respond to the *write*( ) function to transfer data to the device or the *read*( ) function to collect information from the device. The interpretation of the information is defined by the implementor of the driver.

The term *special device* refers to hardware, or an object that appears to the application as such; access to the driver for this hardware uses the file abstraction *character special file*. Implementations supporting the Device Control option shall provide the means to integrate a device driver into the system. The means available to integrate drivers into the system and the way character special files that refer to them are created are implementation defined. Character special files that have no structure defined by this standard can be accessed using the *posix_devctl*( ) function defined in the System Interfaces volume of POSIX.1-2024.

4.22 Thread-Safety

Refer to XSH Section 2.9 (on page 537).

4.23 Treatment of Error Conditions for Mathematical Functions

For all the functions in the **<math.h>** header, an application wishing to check for error situations should set *errno* to 0 and call *feclearexcept*(FE_ALL_EXCEPT) before calling the function. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

On implementations that support the IEC 60559 Floating-Point option, whether or when functions in the **<math.h>** header raise an undeserved underflow floating-point exception is unspecified. Otherwise, as implied by XSH *feraiseexcept*(), the **<math.h>** functions do not raise spurious floating-point exceptions (detectable by the user), other than the inexact floating-point exception.

The error conditions defined for all functions in the **<math.h>** header are domain, pole and range errors, described below. If a domain, pole, or range error occurs and the integer expression (math_errhandling & MATH_ERRNO) is zero, then *errno* shall either be set to the value corresponding to the error, as specified below, or be left unmodified. If no such error occurs, *errno* shall be left unmodified regardless of the setting of *math_errhandling*.

4.23.1 Domain Error

A ``domain error'' shall occur if an input argument is outside the domain over which the mathematical function is defined. The description of each function lists any required domain errors; an implementation may define additional domain errors, provided that such errors are consistent with the mathematical definition of the function.

On a domain error, the function shall return an implementation-defined value; if the integer expression (math_errhandling & MATH_ERRNO) is non-zero, *errno* shall be set to [EDOM]; if the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, the ``invalid'' floating-point exception shall be raised.

4.23.2 Pole Error

A ``pole error'' shall occur if the mathematical result of the function has an exact infinite result as the finite input argument(s) are approached in the limit (for example, `log(0.0)`). The description of each function lists any required pole errors; an implementation may define additional pole errors, provided that such errors are consistent with the mathematical definition of the function.

On a pole error, the function shall return the value of the macro HUGE_VAL, HUGE_VALF, or HUGE_VALL according to the return type, with the same sign as the correct value of the function; if the integer expression (math_errhandling & MATH_ERRNO) is non-zero, *errno* shall be set to [ERANGE]; if the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, the ``divide-by-zero'' floating-point exception shall be raised.

4.23.3 Range Error

A ``range error'' shall occur if the finite mathematical result of the function cannot be represented in an object of the specified type, due to extreme magnitude. The description of each function lists any required range errors; an implementation may define additional range errors, provided that such errors are consistent with the mathematical definition of the function and are the result of either overflow or underflow.

4.23.3.1 Result Overflows

A floating result overflows if the magnitude of the mathematical result is finite but so large that the mathematical result cannot be represented without extraordinary roundoff error in an object of the specified type. If a floating result overflows and default rounding is in effect, then the function shall return the value of the macro HUGE_VAL, HUGE_VALF, or HUGE_VALL according to the return type, with the same sign as the correct value of the function; if the integer expression (math_errhandling & MATH_ERRNO) is non-zero, *errno* shall be set to [ERANGE]; if the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, the ``overflow'' floating-point exception shall be raised.

4.23.3.2 Result Underflows

The result underflows if the magnitude of the mathematical result is so small that the mathematical result cannot be represented, without extraordinary roundoff error, in an object of the specified type. If the result underflows, the function shall return an implementation-defined value whose magnitude is no greater than the smallest normalized positive number in the specified type; if the integer expression (math_errhandling & MATH_ERRNO) is non-zero, whether *errno* is set to [ERANGE] is implementation-defined; if the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, whether the ``underflow'' floating-point exception is raised is implementation-defined.

4.24 Treatment of NaN Arguments for the Mathematical Functions

For functions called with a NaN argument, no errors shall occur and a NaN shall be returned, except where stated otherwise.

If a function with one or more NaN arguments returns a NaN result, the result should be the same as one of the NaN arguments (after possible type conversion), except perhaps for the sign.

On implementations that support the IEC 60559:1989 standard floating point, functions with signaling NaN argument(s) shall be treated as if the function were called with an argument that is a required domain error and shall return a quiet NaN result, except where stated otherwise.

Note: The function might never see the signaling NaN, since it might trigger when the arguments are evaluated during the function call.

On implementations that support the IEC 60559:1989 standard floating point, for those functions that do not have a documented domain error, the following shall apply:

These functions shall fail if:

Domain Error Any argument is a signaling NaN.

Either, the integer expression (math_errhandling & MATH_ERRNO) is non-zero and *errno* shall be set to [EDOM], or the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero and the invalid floating-point exception shall be raised.

4.25 Utility

A utility program shall be either an executable file, such as might be produced by a compiler or linker system from computer source code, or a file of shell source code, directly interpreted by the shell. The program may have been produced by the user, provided by the system implementor, or acquired from an independent distributor.

The system may implement certain utilities as shell functions (see XCU Section 2.9.5, on page 2511) or built-in utilities, but only an application that is aware of the command search order (as described in XCU Section 2.9.1.4, on page 2502) or of performance characteristics can discern differences between the behavior of such a function or built-in utility and that of an executable file.

4.26 Variable Assignment

In the shell command language, a word consisting of the following parts:

```
varname=value
```

When used in a context where assignment is defined to occur and at no other time, the *value* (representing a word or field) shall be assigned as the value of the variable denoted by *varname*. Assignment context occurs in the *cmd_prefix* portion of a shell simple command, as well as in arguments of a recognized declaration utility.

Note: For further information, see XCU Section 2.9.1 (on page 2500).

The *varname* and *value* parts shall meet the requirements for a name and a word, respectively, except that they are delimited by the embedded unquoted <equals-sign>, in addition to other delimiters.

Note: Additional delimiters are described in XCU Section 2.3 (on page 2475).

When a variable assignment is done, the variable shall be created if it did not already exist. If *value* is not specified, the variable shall be given a null value.

Note: An alternative form of variable assignment:

```
symbol=value
```

(where *symbol* is a valid word delimited by an <equals-sign>, but not a valid name) produces unspecified results. The form *symbol=value* is used by the KornShell *name*[*expression*]=*value* syntax.

Chapter 5

File Format Notation

The STDIN, STDOUT, STDERR, INPUT FILES, and OUTPUT FILES sections of the utility descriptions use a syntax to describe the data organization within the files, when that organization is not otherwise obvious. The syntax is similar to that used by the System Interfaces volume of POSIX.1-2024 *printf*( ) function, as described in this chapter. When used in STDIN or INPUT FILES sections of the utility descriptions, this syntax describes the format that could have been used to write the text to be read, not a format that could be used by the System Interfaces volume of POSIX.1-2024 *scanf*( ) function to read the input file.

The description of an individual record is as follows:

"<*format*>**", [**<*arg1*>**,** <*arg2*>**,..., **<*argn*>**]**

The *format* is a character string that contains three types of objects defined below:

1. *Characters* that are not ``escape sequences'' or ``conversion specifications'', as described below, shall be copied to the output.
2. *Escape Sequences* represent non-graphic characters and the escape character (<backslash>).
3. *Conversion Specifications* specify the output format of each argument; see below.

The following characters have the following special meaning in the format string:

' ' (An empty character position.) Represents one or more <blank> characters from the portable character set.

Δ Represents exactly one <space> character.

Table 5-1 lists escape sequences and associated actions on display devices capable of the action.

Table 5-1 Escape Sequences and Associated Actions

Escape Sequence	Represents Character	Terminal Action
\\	<backslash>	Print the <backslash> character.
\a	<alert>	Attempt to alert the user through audible or visible notification.
\b	<backspace>	Move the printing position to one column before the current position, unless the current position is the start of a line.
\f	<form-feed>	Move the printing position to the initial printing position of the next logical page.
\n	<newline>	Move the printing position to the start of the next line.
\r	<carriage-return>	Move the printing position to the start of the current line.
\t	<tab>	Move the printing position to the next tab position on the current line. If there are no more tab positions remaining on the line, the behavior is undefined.
\v	<vertical-tab>	Move the printing position to the start of the next <vertical-tab> position. If there are no more <vertical-tab> positions left on the page, the behavior is undefined.

Each conversion specification is introduced by the <percent-sign> character ('%'). After the character '%', the following shall appear in sequence:

flags Zero or more *flags*, in any order, that modify the meaning of the conversion specification.

field width An optional string of decimal digits to specify a minimum field width. For an output field, if the converted value has fewer bytes than the field width, it shall be padded on the left (or right, if the left-adjustment flag ('−'), described below, has been given) to the field width.

precision Gives the minimum number of digits to appear for the d, o, i, u, x, or X conversion specifiers (the field is padded with leading zeros), the number of digits to appear after the radix character for the e and f conversion specifiers, the maximum number of significant digits for the g conversion specifier; or the maximum number of bytes to be written from a string in the s conversion specifier. The precision shall take the form of a <period> ('.') followed by a decimal digit string; a null digit string is treated as zero.

conversion specifier characters
A conversion specifier character (see below) that indicates the type of conversion to be applied.

The *flag* characters and their meanings are:

− The result of the conversion shall be left-justified within the field.

+ The result of a signed conversion shall always begin with a sign ('+' or '−').

<space> If the first character of a signed conversion is not a sign, a <space> shall be prefixed to the result. This means that if the <space> and '+' flags both appear, the <space> flag shall be ignored.

The value shall be converted to an alternative form. For c, d, i, u, and s conversion specifiers, the behavior is undefined. For the o conversion specifier, it shall increase the precision to force the first digit of the result to be a zero. For x or X conversion specifiers, a non-zero result has 0x or 0X prefixed to it, respectively. For a, A, e, E, f, F, g, and G conversion specifiers, the result shall always contain a radix character, even if no digits follow the radix character. For g and G conversion specifiers, trailing zeros shall not be removed from the result as they usually are.

0 For a, A, d, e, E, f, F, g, G, i, o, u, x, and X conversion specifiers, leading zeros (following any indication of sign or base) shall be used to pad to the field width rather than performing space padding, except when converting an infinity or NaN. If the '0' and '−' flags both appear, the '0' flag shall be ignored. For d, i, o, u, x, and X conversion specifiers, if a precision is specified, the '0' flag shall be ignored. For other conversion specifiers, the behavior is undefined.

Each conversion specifier character shall result in fetching zero or more arguments. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments shall be ignored.

The conversion specifiers and their meanings are:

a,A The floating-point number argument representing a floating-point number shall be converted in the style "[−]0x*h*.*hhhh*p±*d*", where there is one hexadecimal digit (which shall be non-zero if the argument is a normalized floating-point number and is otherwise unspecified) before the decimal-point character and the number of hexadecimal digits after it is equal to the precision; if the precision is missing

and FLT_RADIX is a power of 2, then the precision shall be sufficient for an exact representation of the value; if the precision is missing and FLT_RADIX is not a power of 2, then the precision shall be sufficient to distinguish different floating-point values in the internal representation used by the utility, except that trailing zeros may be omitted; if the precision is zero and the # flag is not specified, no decimal-point character shall appear. The letters "abcdef" shall be used for a conversion and the letters "ABCDEF" for A conversion. The A conversion specifier produces a number with X and P instead of x and p. The exponent shall always contain at least one digit, and only as many more digits as necessary to represent the decimal exponent of 2. If the value is zero, the exponent shall be zero. A floating-point number argument representing an infinity or NaN shall be converted in the style of an f or F conversion specifier.

d,i,o,u,x,X — The integer argument shall be written as signed decimal (d or i), unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal notation (x and X). The d and i specifiers shall convert to signed decimal in the style "[−]*dddd*". The x conversion specifier shall use the numbers and letters "0123456789abcdef" and the X conversion specifier shall use the numbers and letters "0123456789ABCDEF". The *precision* component of the argument shall specify the minimum number of digits to appear. If the value being converted can be represented in fewer digits than the specified minimum, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting a zero value with a precision of 0 shall be no characters. If both the field width and precision are omitted, the implementation may precede, follow, or precede and follow numeric arguments of types d, i, and u with <blank> characters from the portable character set; arguments of type o (octal) may be preceded with leading zeros.

f,F — The floating-point number argument shall be written in decimal notation in the style [−]*ddd.ddd*, where the number of digits after the radix character (shown here as a decimal point) shall be equal to the *precision* specification. The *LC_NUMERIC* locale category shall determine the radix character to use in this format. If the *precision* is omitted from the argument, six digits shall be written after the radix character; if the *precision* is explicitly 0, no radix character shall appear.

A floating-point number argument representing an infinity shall be converted in one of the styles "[−]inf" or "[−]infinity"; which style is implementation-defined. A floating-point number argument representing a NaN shall be converted in one of the styles "[−]nan(*n-char-sequence*)" or "[−]nan"; which style, and the meaning of any *n-char-sequence*, is implementation-defined. The F conversion specifier produces "INF", "INFINITY", or "NAN" instead of "inf", "infinity", or "nan", respectively.

e,E — The floating-point number argument shall be written in the style [−]*d.ddd*±*dd* (the symbol '±' indicates either a <plus-sign> or <hyphen-minus>), where there is one digit before the radix character (shown here as a decimal point) and the number of digits after it is equal to the precision. The *LC_NUMERIC* locale category shall determine the radix character to use in this format. When the precision is missing, six digits shall be written after the radix character; if the precision is 0, no radix character shall appear. The E conversion specifier shall produce a number with E instead of e introducing the exponent. The exponent shall always contain at least two digits. However, if the value to be written requires an exponent greater than two digits, additional exponent digits shall be written as necessary.

A floating-point number argument representing an infinity or NaN shall be

converted in the style of an `f` or `F` conversion specifier.

`g,G` The floating-point number argument shall be written in style `f` or `e` (or in style `F` or `E` in the case of a `G` conversion specifier), with the precision specifying the number of significant digits. The style used depends on the value converted: style `e` (or `E`) shall be used only if the exponent resulting from the conversion is less than –4 or greater than or equal to the precision. Trailing zeros are removed from the result. A radix character shall appear only if it is followed by a digit.

A floating-point number argument representing an infinity or NaN shall be converted in the style of an `f` or `F` conversion specifier.

`c` The single-byte character argument shall be written.

`s` The argument shall be taken to be a string and bytes from the string shall be written until the end of the string or the number of bytes indicated by the *precision* specification of the argument is reached. If the precision is omitted from the argument, it shall be taken to be infinite, so all bytes up to the end of the string shall be written.

`%` Write a `'%'` character; no argument shall be converted. Applications using the *printf* utility shall ensure that the complete conversion specification is `%%`.

In no case does a nonexistent or insufficient field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. The term ``field width'' should not be confused with the term ``precision'' used in the description of `%s`.

Examples

To represent the output of a program that prints a date and time in the form Sunday, July 3, 10:02, where *weekday* and *month* are strings:

```
"%s,Δ%sΔ%d,Δ%d:%.2d\n" <weekday>, <month>, <day>, <hour>, <min>
```

To show 'π' written to 5 decimal places:

```
"piΔ=Δ%.5f\n",<value of π>
```

To show an input file format consisting of five <colon>-separated fields:

```
"%s:%s:%s:%s:%s\n", <arg1>, <arg2>, <arg3>, <arg4>, <arg5>
```


Chapter 6
Character Set

6.1 Portable Character Set

Conforming implementations shall support one or more coded character sets. Each supported locale shall include the *portable character set*, which is the set of symbolic names for characters in Table 6-1. This is used to describe characters within the text of POSIX.1-2024. The first eight entries in Table 6-1 and all characters in Table 6-2 (on page 122) are defined in the ISO/IEC 6429:1992 standard. The rest of the characters in Table 6-1 are defined in the ISO/IEC 10646:2020 standard.

Table 6-1 Portable Character Set

Symbolic Name(s)	Glyph	UCS	Description
<NUL>		<U0000>	NULL (NUL)
<alert>, <BEL>		<U0007>	BELL
<backspace>, <BS>		<U0008>	BACKSPACE
<tab>, <HT>		<U0009>	CHARACTER TABULATION
<newline>, <LF>		<U000A>	LINE FEED (LF)
<vertical-tab>, <VT>		<U000B>	LINE TABULATION
<form-feed>, <FF>		<U000C>	FORM FEED (FF)
<carriage-return>, <CR>		<U000D>	CARRIAGE RETURN (CR)
<space>		<U0020>	SPACE
<exclamation-mark>	!	<U0021>	EXCLAMATION MARK
<quotation-mark>	"	<U0022>	QUOTATION MARK
<number-sign>	#	<U0023>	NUMBER SIGN
<dollar-sign>	$	<U0024>	DOLLAR SIGN
<percent-sign>	%	<U0025>	PERCENT SIGN
<ampersand>	&	<U0026>	AMPERSAND
<apostrophe>	'	<U0027>	APOSTROPHE
<left-parenthesis>	(	<U0028>	LEFT PARENTHESIS
<right-parenthesis>	)	<U0029>	RIGHT PARENTHESIS
<asterisk>	*	<U002A>	ASTERISK
<plus-sign>	+	<U002B>	PLUS SIGN
<comma>	,	<U002C>	COMMA
<hyphen-minus>, <hyphen>	–	<U002D>	HYPHEN-MINUS
<full-stop>, <period>	.	<U002E>	FULL STOP
<slash>, <solidus>	/	<U002F>	SOLIDUS
<zero>	0	<U0030>	DIGIT ZERO
<one>	1	<U0031>	DIGIT ONE
<two>	2	<U0032>	DIGIT TWO
<three>	3	<U0033>	DIGIT THREE
<four>	4	<U0034>	DIGIT FOUR
<five>	5	<U0035>	DIGIT FIVE

Symbolic Name(s)	Glyph	UCS	Description
<six>	6	<U0036>	DIGIT SIX
<seven>	7	<U0037>	DIGIT SEVEN
<eight>	8	<U0038>	DIGIT EIGHT
<nine>	9	<U0039>	DIGIT NINE
<colon>	:	<U003A>	COLON
<semicolon>	;	<U003B>	SEMICOLON
<less-than-sign>	<	<U003C>	LESS-THAN SIGN
<equals-sign>	=	<U003D>	EQUALS SIGN
<greater-than-sign>	>	<U003E>	GREATER-THAN SIGN
<question-mark>	?	<U003F>	QUESTION MARK
<commercial-at>	@	<U0040>	COMMERCIAL AT
<A>	A	<U0041>	LATIN CAPITAL LETTER A
<B>	B	<U0042>	LATIN CAPITAL LETTER B
<C>	C	<U0043>	LATIN CAPITAL LETTER C
<D>	D	<U0044>	LATIN CAPITAL LETTER D
<E>	E	<U0045>	LATIN CAPITAL LETTER E
<F>	F	<U0046>	LATIN CAPITAL LETTER F
<G>	G	<U0047>	LATIN CAPITAL LETTER G
<H>	H	<U0048>	LATIN CAPITAL LETTER H
<I>	I	<U0049>	LATIN CAPITAL LETTER I
<J>	J	<U004A>	LATIN CAPITAL LETTER J
<K>	K	<U004B>	LATIN CAPITAL LETTER K
<L>	L	<U004C>	LATIN CAPITAL LETTER L
<M>	M	<U004D>	LATIN CAPITAL LETTER M
<N>	N	<U004E>	LATIN CAPITAL LETTER N
<O>	O	<U004F>	LATIN CAPITAL LETTER O
<P>	P	<U0050>	LATIN CAPITAL LETTER P
<Q>	Q	<U0051>	LATIN CAPITAL LETTER Q
<R>	R	<U0052>	LATIN CAPITAL LETTER R
<S>	S	<U0053>	LATIN CAPITAL LETTER S
<T>	T	<U0054>	LATIN CAPITAL LETTER T
<U>	U	<U0055>	LATIN CAPITAL LETTER U
<V>	V	<U0056>	LATIN CAPITAL LETTER V
<W>	W	<U0057>	LATIN CAPITAL LETTER W
<X>	X	<U0058>	LATIN CAPITAL LETTER X
<Y>	Y	<U0059>	LATIN CAPITAL LETTER Y
<Z>	Z	<U005A>	LATIN CAPITAL LETTER Z
<left-square-bracket>	[	<U005B>	LEFT SQUARE BRACKET
<backslash>, <reverse-solidus>	\	<U005C>	REVERSE SOLIDUS
<right-square-bracket>	]	<U005D>	RIGHT SQUARE BRACKET
<circumflex-accent>, <circumflex>	^	<U005E>	CIRCUMFLEX ACCENT
<low-line>, <underscore>	_	<U005F>	LOW LINE
<grave-accent>	`	<U0060>	GRAVE ACCENT
<a>	a	<U0061>	LATIN SMALL LETTER A
<b>	b	<U0062>	LATIN SMALL LETTER B
<c>	c	<U0063>	LATIN SMALL LETTER C
<d>	d	<U0064>	LATIN SMALL LETTER D
<e>	e	<U0065>	LATIN SMALL LETTER E
<f>	f	<U0066>	LATIN SMALL LETTER F
<g>	g	<U0067>	LATIN SMALL LETTER G
<h>	h	<U0068>	LATIN SMALL LETTER H

Symbolic Name(s)	Glyph	UCS	Description
<i>	i	<U0069>	LATIN SMALL LETTER I
<j>	j	<U006A>	LATIN SMALL LETTER J
<k>	k	<U006B>	LATIN SMALL LETTER K
<l>	l	<U006C>	LATIN SMALL LETTER L
<m>	m	<U006D>	LATIN SMALL LETTER M
<n>	n	<U006E>	LATIN SMALL LETTER N
<o>	o	<U006F>	LATIN SMALL LETTER O
<p>	p	<U0070>	LATIN SMALL LETTER P
<q>	q	<U0071>	LATIN SMALL LETTER Q
<r>	r	<U0072>	LATIN SMALL LETTER R
<s>	s	<U0073>	LATIN SMALL LETTER S
<t>	t	<U0074>	LATIN SMALL LETTER T
<u>	u	<U0075>	LATIN SMALL LETTER U
<v>	v	<U0076>	LATIN SMALL LETTER V
<w>	w	<U0077>	LATIN SMALL LETTER W
<x>	x	<U0078>	LATIN SMALL LETTER X
<y>	y	<U0079>	LATIN SMALL LETTER Y
<z>	z	<U007A>	LATIN SMALL LETTER Z
<left-brace>, <left-curly-bracket>	{	<U007B>	LEFT CURLY BRACKET
<vertical-line>	|	<U007C>	VERTICAL LINE
<right-brace>, <right-curly-bracket>	}	<U007D>	RIGHT CURLY BRACKET
<tilde>	~	<U007E>	TILDE

POSIX.1-2024 uses character names other than the above, but only in an informative way; for example, in examples to illustrate the use of characters beyond the portable character set with the facilities of POSIX.1-2024.

Table 6-1 (on page 117) defines the characters in the portable character set and the corresponding symbolic character names used to identify each character in a character set description file. Characters defined in Table 6-2 (on page 122) may also be used in character set description files.

POSIX.1-2024 places only the following requirements on the encoded values of the characters in the portable character set:

- If the encoded values associated with each member of the portable character set are not invariant across all locales supported by the implementation, if an application uses any pair of locales where the character encodings differ, or accesses data from an application using a locale which has different encodings from the locales used by the application, the results are unspecified.
- The encoded values associated with the digits 0 to 9 shall be such that the value of each character after 0 shall be one greater than the value of the previous character.
- A null character, NUL, which has all bits set to zero, shall be in the set of characters.
- The encoded values associated with <period>, <slash>, <newline>, and <carriage-return> shall be invariant across all locales supported by the implementation.
- The encoded values associated with the members of the portable character set are each represented in a single byte. Moreover, if the value is stored in an object of C-language type **char**, it is guaranteed to be positive (except the NUL, which is always zero).

Conforming implementations shall support certain character and character set attributes, as defined in Section 7.2 (on page 128).

6.2 Character Encoding

The POSIX locale shall contain 256 single-byte characters including the characters in Table 6-1 (on page 117) and Table 6-2 (on page 122), which have the properties listed in Section 7.3.1 (on page 131). It is unspecified whether characters not listed in those two tables are classified as **punct** or **cntrl**, or neither. Other locales shall contain the characters in Table 6-1 (on page 117) and may contain any or all of the control characters identified in Table 6-2 (on page 122); the presence, meaning, and representation of any additional characters are locale-specific.

In locales other than the POSIX locale, a character may have a state-dependent encoding. There are two types of these encodings:

- A single-shift encoding (where each character not in the initial shift state is preceded by a shift code) can be defined if each shift-code and character sequence is considered a multi-byte character. This is done using the concatenated-constant format in a character set description file, as described in Section 6.4 (on page 121). If the implementation supports a character encoding of this type, all of the standard utilities in the Shell and Utilities volume of POSIX.1-2024 shall support it. Use of a single-shift encoding with any of the functions in the System Interfaces volume of POSIX.1-2024 that do not specifically mention the effects of state-dependent encoding is implementation-defined.
- A locking-shift encoding (where the state of the character is determined by a shift code that may affect more than the single character following it) cannot be defined with the current character set description file format. Use of a locking-shift encoding with any of the standard utilities in the Shell and Utilities volume of POSIX.1-2024 or with any of the functions in the System Interfaces volume of POSIX.1-2024 that do not specifically mention the effects of state-dependent encoding is implementation-defined.

While in the initial shift state, all characters in the portable character set shall retain their usual interpretation and shall not alter the shift state. The interpretation for subsequent bytes in the sequence shall be a function of the current shift state. A byte with all bits zero shall be interpreted as the null character independent of shift state. Such a byte shall not occur as part of any other character. Likewise, the byte values used to encode <period>, <slash>, <newline>, and <carriage-return> shall not occur as part of any other character in any locale.

The maximum allowable number of bytes in a character in the current locale shall be indicated by {MB_CUR_MAX}, defined in the **<stdlib.h>** header and by the **<mb_cur_max>** value in a character set description file; see Section 6.4 (on page 121). The implementation's maximum number of bytes in a character shall be defined by the C-language macro {MB_LEN_MAX}.

6.3 C Language Wide-Character Codes

In the shell, the standard utilities are written so that the encodings of characters are described by the locale's *LC_CTYPE* definition (see Section 7.3.1, on page 131) and there is no differentiation between characters consisting of single octets (8-bit bytes) or multiple bytes. However, in the C language, a differentiation is made. To ease the handling of variable length characters, the C language has introduced the concept of wide-character codes.

All wide-character codes in a given process consist of an equal number of bits. This is in contrast to characters, which can consist of a variable number of bytes. The byte or byte sequence that represents a character can also be represented as a wide-character code. Wide-character codes thus provide a uniform size for manipulating text data. A wide-character code having all bits zero is the null wide-character code (see Section 3.233, on page 65), and terminates wide-character strings (see Section 3.417, on page 93). The wide-character value for each member of the portable character set shall equal its value when used as the lone character in an integer

character constant. Wide-character codes for other characters are locale and implementation-defined. State shift bytes shall not have a wide-character code representation. POSIX.1-2024 provides no means of defining a wide-character codeset.

Arguments to the functions declared in the **<wchar.h>** header can point to arrays containing **wchar_t** values that do not correspond to valid wide character codes according to the *LC_CTYPE* category of the locale being used. Such values shall be processed according to the specified semantics for the function in the System Interfaces volume of POSIX.1-2024, except that it is unspecified whether an encoding error occurs if such a value appears in the format string of a function that has a format string as a parameter and the specified semantics do not require that value to be processed as if by *wcrtomb*( ).

6.4 Character Set Description File

Implementations shall provide a character set description file for at least one coded character set supported by the implementation. These files are referred to elsewhere in POSIX.1-2024 as *charmap* files. It is implementation-defined whether or not users or applications can provide additional character set description files.

POSIX.1-2024 does not require that multiple character sets or codesets be supported. Although multiple charmap files are supported, it is the responsibility of the implementation to provide the file or files; if only one is provided, only that one is accessible using the *localedef* utility's **–f** option.

Each character set description file, except those that use the ISO/IEC 10646:2020 standard position values as the encoding values, shall define characteristics for the coded character set and the encoding for the characters specified in Table 6-1 (on page 117), and may define encoding for additional characters supported by the implementation. Other information about the coded character set may also be in the file. Coded character set character values shall be defined using symbolic character names followed by character encoding values.

Each symbolic name specified in Table 6-1 (on page 117) shall be included in the file. Each character in Table 6-1 (on page 117) (each row in the table) shall be mapped to a unique coding value. For each character in Table 6-2 (on page 122) that exists in the character set described by the file, the character's symbolic name(s) from Table 6-2 (on page 122) and the character's single-byte encoding value shall be included in the file.

Table 6-2 Non-Portable Control Characters

Symbolic Name(s)	UCS	Description
<SOH>	<U0001>	START OF HEADING
<STX>	<U0002>	START OF TEXT
<ETX>	<U0003>	END OF TEXT
<EOT>	<U0004>	END OF TRANSMISSION
<ENQ>	<U0005>	ENQUIRY
<ACK>	<U0006>	ACKNOWLEDGE
<SO>	<U000E>	SHIFT OUT
<SI>	<U000F>	SHIFT IN
<DLE>	<U0010>	DATA LINK ESCAPE
<DC1>	<U0011>	DEVICE CONTROL ONE
<DC2>	<U0012>	DEVICE CONTROL TWO
<DC3>	<U0013>	DEVICE CONTROL THREE
<DC4>	<U0014>	DEVICE CONTROL FOUR
<NAK>	<U0015>	NEGATIVE ACKNOWLEDGE
<SYN>	<U0016>	SYNCHRONOUS IDLE
<ETB>	<U0017>	END OF TRANSMISSION BLOCK
<CAN>	<U0018>	CANCEL
<EM>	<U0019>	END OF MEDIUM
<SUB>	<U001A>	SUBSTITUTE
<ESC>	<U001B>	ESCAPE
<IS4>, <FS>	<U001C>	INFORMATION SEPARATOR FOUR
<IS3>, <GS>	<U001D>	INFORMATION SEPARATOR THREE
<IS2>, <RS>	<U001E>	INFORMATION SEPARATOR TWO
<IS1>, <US>	<U001F>	INFORMATION SEPARATOR ONE
<DEL>	<U007F>	DELETE

The following declarations can precede the character definitions. Each shall consist of the symbol shown in the following list, starting in column 1, including the surrounding brackets, followed by one or more <blank> characters, followed by the value to be assigned to the symbol.

<code_set_name> The name of the coded character set for which the character set description file is defined. The characters of the name shall be taken from the set of characters with visible glyphs defined in Table 6-1 (on page 117).

<mb_cur_max> The maximum number of bytes in a multi-byte character. This shall default to 1.

<mb_cur_min> XSI An unsigned positive integer value that defines the minimum number of bytes in a character for the encoded character set. On XSI-conformant systems, **<mb_cur_min>** shall always be 1.

<escape_char> The character used to indicate that the characters following shall be interpreted in a special way, as defined later in this section. This shall default to <backslash> (`'\\'`), which is the character used in all the following text and examples, unless otherwise noted.

<comment_char> The character that, when placed in column 1 of a charmap line, is used to indicate that the line shall be ignored. The default character shall be the <number-sign> (`'#'`).

The character set mapping definitions shall be all the lines immediately following an identifier line containing the string `"CHARMAP"` starting in column 1, and preceding a trailer line

containing the string "END CHARMAP" starting in column 1. Empty lines and lines containing a **<comment_char>** in the first column shall be ignored. Each non-comment line of the character set mapping definition (that is, between the "CHARMAP" and "END CHARMAP" lines of the file) shall be in either of two forms:

"%s %s %s\n", <*symbolic-name*>, <*encoding*>, <*comments*>

or:

"%s...%s %s %s\n", <*symbolic-name*>, <*symbolic-name*>,
    <*encoding*>, <*comments*>

In the first format, the line in the character set mapping definition shall define a single symbolic name and a corresponding encoding. A symbolic name is one or more characters from the set shown with visible glyphs in Table 6-1 (on page 117), enclosed between angle brackets. A character following an escape character is interpreted as itself; for example, the sequence "<\\\>>" represents the symbolic name "\>" enclosed between angle brackets.

In the second format, the line in the character set mapping definition shall define a range of one or more symbolic names. In this form, the symbolic names shall consist of zero or more non-numeric characters from the set shown with visible glyphs in Table 6-1 (on page 117), followed by an integer formed by one or more decimal digits. Both integers shall contain the same number of digits. The characters preceding the integer shall be identical in the two symbolic names, and the integer formed by the digits in the second symbolic name shall be equal to or greater than the integer formed by the digits in the first name. This shall be interpreted as a series of symbolic names formed from the common part and each of the integers between the first and the second integer, inclusive. As an example, <j0101>...<j0104> is interpreted as the symbolic names <j0101>, <j0102>, <j0103>, and <j0104>, in that order.

A character set mapping definition line shall exist for all symbolic names specified in Table 6-1 (on page 117), and shall define the coded character value that corresponds to the character indicated in the table, or the coded character value that corresponds to the control character symbolic name. If the control characters commonly associated with the symbolic names in Table 6-2 (on page 122) are supported by the implementation, the symbolic name and the corresponding encoding value shall be included in the file. Additional unique symbolic names may be included. A coded character value can be represented by more than one symbolic name.

The encoding part is expressed as one (for single-byte character values) or more concatenated decimal, octal, or hexadecimal constants in the following formats:

"%cd%u", <*escape_char*>, <*decimal byte value*>
"%cx%x", <*escape_char*>, <*hexadecimal byte value*>
"%c%o", <*escape_char*>, <*octal byte value*>

Decimal constants shall be represented by two or three decimal digits, preceded by the escape character and the lowercase letter 'd'; for example, "\d05", "\d97", or "\d143". Hexadecimal constants shall be represented by two hexadecimal digits, preceded by the escape character and the lowercase letter 'x'; for example, "\x05", "\x61", or "\x8f". Octal constants shall be represented by two or three octal digits, preceded by the escape character; for example, "\05", "\141", or "\217". In a portable charmap file, each constant represents an 8-bit byte. When constants are concatenated for multi-byte character values, they shall be of the same type, and interpreted in sequence from from first to last with the first byte of the multi-byte character specified by the first byte in the sequence. The manner in which these constants are represented in the character stored in the system is implementation-defined. (This notation was chosen for reasons of portability. There is no requirement that the internal representation in the computer memory be in this same order.) Omitting bytes from a multi-byte character definition produces undefined results.

In lines defining ranges of symbolic names, the encoded value shall be the value for the first symbolic name in the range (the symbolic name preceding the ellipsis). Subsequent symbolic names defined by the range shall have encoding values in increasing order. Bytes shall be treated as unsigned octets, and carry shall be propagated between the bytes as necessary to represent the range. However, because this causes a null byte in the second or subsequent bytes of a character, such a declaration should not be specified. For example, the line:

```
<j0101>...<j0104>  \d129\d254
```

is interpreted as:

```
<j0101>            \d129\d254
<j0102>            \d129\d255
<j0103>            \d130\d00
<j0104>            \d130\d01
```

The expanded declaration of the symbol **<**j0103**>** in the above example is an invalid specification, because it contains a null byte in the second byte of a character.

The comment is optional.

POSIX.1-2024 provides no means of defining a wide-character codeset.

The following declarations can follow the character set mapping definitions (after the "`END CHARMAP`" statement). Each shall consist of the keyword shown in the following list, starting in column 1, followed by the value(s) to be associated to the keyword, as defined below.

WIDTH A non-negative integer value defining the column width (see Section 3.75, on page 42) for the printable characters in the coded character set specified in Table 6-1 (on page 117) and Table 6-2 (on page 122). Coded character set character values shall be defined using symbolic character names followed by column width values. Defining a character with more than one **WIDTH** produces undefined results. The **END WIDTH** keyword shall be used to terminate the **WIDTH** definitions. Specifying the width of a non-printable character in a **WIDTH** declaration produces undefined results.

WIDTH_DEFAULT
A non-negative integer value defining the default column width for any printable character not listed by one of the **WIDTH** keywords. If no **WIDTH_DEFAULT** keyword is included in the charmap, the default character width shall be 1.

Example

After the "`END CHARMAP`" statement, a syntax for a width definition would be:

```
WIDTH
<A> 1
<B> 1
<C>...<Z> 1
...
<foo1>...<foon> 2
...
END WIDTH
```

In this example, the numerical code point values represented by the symbols **<A>** and **<B>** are assigned a width of 1. The code point values **<C>** to **<Z>** inclusive (**<C>**, **<D>**, **<E>**, and so on) are also assigned a width of 1. Using **<A>**...**<Z>** would have required fewer lines, but the alternative was shown to demonstrate flexibility. The keyword **WIDTH_DEFAULT** could have

been added as appropriate.

6.4.1 State-Dependent Character Encodings

This section addresses the use of state-dependent character encodings (that is, those in which the encoding of a character is dependent on one or more shift codes that may precede it).

A single-shift encoding (where each character not in the initial shift state is preceded by a shift code) can be defined in the charmap format if each shift-code/character sequence is considered a multi-byte character, defined using the concatenated-constant format described in Section 6.4 (on page 121). If the implementation supports a character encoding of this type, all of the standard utilities shall support it. A locking-shift encoding (where the state of the character is determined by a shift code that may affect more than the single character following it) could be defined with an extension to the charmap format described in Section 6.4 (on page 121).

If the implementation supports a character encoding of this type, any of the standard utilities that describe character (*versus* byte) or text-file manipulation shall have the following characteristics:

1. The utility shall process the statefully encoded data as a concatenation of state-independent characters. The presence of redundant locking shifts shall not affect the comparison of two statefully encoded strings.
2. A utility that divides, truncates, or extracts substrings from statefully encoded data shall produce output that contains locking shifts at the beginning or end of the resulting data, if appropriate, to retain correct state information.

Chapter 7
Locale

7.1 General

A locale is the definition of the subset of a user's environment that depends on language and cultural conventions. It is made up from one or more categories. Each category is identified by its name and controls specific aspects of the behavior of components of the system. Category names correspond to the following environment variable names:

LC_CTYPE Character classification and case conversion.

LC_COLLATE Collation order.

LC_MONETARY Monetary formatting.

LC_NUMERIC Numeric, non-monetary formatting.

LC_TIME Date and time formats.

LC_MESSAGES Formats of informative and diagnostic messages and interactive responses.

The standard utilities in the Shell and Utilities volume of POSIX.1-2024 shall base their behavior on the current locale, as defined in the ENVIRONMENT VARIABLES section for each utility. The behavior of some of the C-language functions defined in the System Interfaces volume of POSIX.1-2024 shall also be modified based on a locale selection. The locale to be used by these functions can be selected in the following ways:

1. For functions such as *isalnum_l*( ) that take a locale object as an argument, a locale object can be obtained from *newlocale*( ) or *duplocale*( ) and passed to the function.
2. For functions that do not take a locale object as an argument, the current locale for the thread can be set by calling *uselocale*( ) or the global locale for the process can be set by calling *setlocale*( ). Such functions shall use the current locale of the calling thread if one has been set for that thread; otherwise, they shall use the global locale.
3. Some functions, such as *catopen*( ) and those related to text domains, may reference various environment variables and a locale category of a specific locale to access files they need to use.

Locales other than those supplied by the implementation can be created via the *localedef* utility, provided that the _POSIX2_LOCALEDEF symbol is defined on the system. Even if *localedef* is not provided, all implementations conforming to the System Interfaces volume of POSIX.1-2024 shall provide one or more locales that behave as described in this chapter. The input to the utility is described in Section 7.3 (on page 128). The value that is used to specify a locale when using environment variables shall be the string specified as the *name* operand to the *localedef* utility when the locale was created. The strings `"C"` and `"POSIX"` are reserved as identifiers for the POSIX locale (see Section 7.2, on page 128). When the value of a locale environment variable begins with a <slash> (`'/'`), it shall be interpreted as the pathname of the locale definition; the type of file (regular, directory, and so on) used to store the locale definition is implementation-defined. If the value does not begin with a <slash>, the mechanism used to locate the locale is implementation-defined.

If incompatible character sets are used by the locale categories, the results achieved by an application utilizing these categories are undefined. Two locale categories have incompatible character sets if one of the categories is *LC_CTYPE* and the locale data associated with the other category includes at least one character that either is not in the character set used by *LC_CTYPE* or has a different encoding than the same character in the character set used by *LC_CTYPE*.

Likewise, unless specified otherwise, if different codesets are used by a particular category of the selected locale and by the data being processed by an interface whose behavior is dependent on that category of the selected locale, the results are undefined.

Applications can select the desired locale by calling the *newlocale*() or *setlocale*() function with the appropriate value. If the function is invoked with an empty string, such as:

```
newlocale(LC_ALL_MASK, "", (locale_t)0);
```

or:

```
setlocale(LC_ALL, "");
```

the value of the corresponding environment variable is used. If the environment variable is unset or is set to the empty string, the implementation shall set the appropriate environment as defined in Chapter 8 (on page 167).

7.2 POSIX Locale

Conforming systems shall provide a POSIX locale, also known as the C locale. In POSIX.1 the requirements for the POSIX locale are more extensive than the requirements for the C locale as specified in the ISO C standard. However, in a conforming POSIX implementation, the POSIX locale and the C locale are identical. The behavior of standard utilities and functions in the POSIX locale shall be as if the locale was defined via the *localedef* utility with input data from the POSIX locale tables in Section 7.3.

For C-language programs, the POSIX locale shall be the default locale when the *setlocale*() function is not called.

The POSIX locale can be specified by assigning to the appropriate environment variables the values `"C"` or `"POSIX"`.

All implementations shall define a locale as the default locale, to be invoked when no environment variables are set, or set to the empty string. This default locale can be the POSIX locale or any other implementation-defined locale. Some implementations may provide facilities for local installation administrators to set the default locale, customizing it for each location. POSIX.1-2024 does not require such a facility.

7.3 Locale Definition

The capability to specify additional locales to those provided by an implementation is optional, denoted by the _POSIX2_LOCALEDEF symbol. If the option is not supported, only implementation-supplied locales are available. Such locales shall be documented using the format specified in this section.

Locales can be described with the file format presented in this section. The file format is that accepted by the *localedef* utility. For the purposes of this section, the file is referred to as the ``locale definition file'', but no locales shall be affected by this file unless it is processed by *localedef* or some similar mechanism. Any requirements in this section imposed upon the utility shall apply to *localedef* or to any other similar utility used to install locale information using the

locale definition file format described here.

The locale definition file shall contain one or more locale category source definitions, and shall not contain more than one definition for the same locale category. If the file contains source definitions for more than one category, implementation-defined categories, if present, shall appear after the categories defined by Section 7.1 (on page 127). A category source definition contains either the definition of a category or a **copy** directive. For a description of the **copy** directive, see *localedef*. In the event that some of the information for a locale category, as specified in this volume of POSIX.1-2024, is missing from the locale source definition, the behavior of that category, if it is referenced, is unspecified.

A category source definition shall consist of a category header, a category body, and a category trailer. A category header shall consist of the character string naming of the category, beginning with the characters *LC_*. The category trailer shall consist of the string `"END"`, followed by one or more <blank> characters and the string used in the corresponding category header.

The category body shall consist of one or more lines of text. Each line shall contain an identifier, optionally followed by one or more operands. Identifiers shall be either keywords, identifying a particular locale element, or collating elements. In addition to the keywords defined in this volume of POSIX.1-2024, the source can contain implementation-defined keywords. Each keyword within a locale shall have a unique name (that is, two categories cannot have a commonly-named keyword); no keyword shall start with the characters *LC_*. Identifiers shall be separated from the operands by one or more <blank> characters.

Operands shall be characters, collating elements, or strings of characters. Strings shall be enclosed in double-quotes. Literal double-quotes within strings shall be preceded by the <*escape character*>, described below. When a keyword is followed by more than one operand, the operands shall be separated by <semicolon> characters; <blank> characters shall be allowed both before and after a <semicolon>.

The first category header in the file can be preceded by a line modifying the comment character. It shall have the following format, starting in column 1:

`"comment_char %c\n", <comment character>`

The comment character shall default to the <number-sign> (`'#'`). Blank lines and lines containing the <*comment character*> in the first position shall be ignored.

The first category header in the file can be preceded by a line modifying the escape character to be used in the file. It shall have the following format, starting in column 1:

`"escape_char %c\n", <escape character>`

The escape character shall default to <backslash>, which is the character used in all examples shown in this volume of POSIX.1-2024.

A line can be continued by placing an escape character as the last character on the line; this continuation character shall be discarded from the input. Although the implementation need not accept any one portion of a continued line with a length exceeding {LINE_MAX} bytes, it shall place no limits on the accumulated length of the continued line. Comment lines shall not be continued on a subsequent line using an escaped <newline>.

Individual characters, characters in strings, and collating elements shall be represented using symbolic names, as defined below. In addition, characters can be represented using the characters themselves or as octal, hexadecimal, or decimal constants. When non-symbolic notation is used, the resultant locale definitions are in many cases not portable between systems. The left angle bracket (`'<'`) is a reserved symbol, denoting the start of a symbolic name; when used to represent itself it shall be preceded by the escape character. The following rules apply to character representation:

1. A character can be represented via a symbolic name, enclosed within angle brackets '<' and '>'. The symbolic name, including the angle brackets, shall exactly match a symbolic name defined in the charmap file specified via the *localedef* **–f** option, and it shall be replaced by a character value determined from the value associated with the symbolic name in the charmap file. The use of a symbolic name not found in the charmap file shall constitute an error, unless the category is *LC_CTYPE* or *LC_COLLATE*, in which case it shall constitute a warning condition (see *localedef* for a description of actions resulting from errors and warnings). The specification of a symbolic name in a **collating-element** or **collating-symbol** section that duplicates a symbolic name in the charmap file (if present) shall be an error. Use of the escape character or a right angle bracket within a symbolic name is invalid unless the character is preceded by the escape character.

   For example:

   ```
   <c>;<c-cedilla>  "<M><a><y>"
   ```

2. A character in the portable character set can be represented by the character itself, in which case the value of the character is implementation-defined. (Implementations may allow other characters to be represented as themselves, but such locale definitions are not portable.) Within a string, the double-quote character, the escape character, and the right angle bracket character shall be escaped (preceded by the escape character) to be interpreted as the character itself. Outside strings, the characters:

   ```
   ,    ;    <    >    escape_char
   ```

   shall be escaped to be interpreted as the character itself.

   For example:

   ```
   c    "May"
   ```

3. A character can be represented as an octal constant. An octal constant shall be specified as the escape character followed by two or three octal digits. Each constant shall represent a byte value. Multi-byte values can be represented by concatenated constants specified in byte order with the last constant specifying the least significant byte of the character.

   For example:

   ```
   \143;\347;\143\150   "\115\141\171"
   ```

4. A character can be represented as a hexadecimal constant. A hexadecimal constant shall be specified as the escape character followed by an 'x' followed by two hexadecimal digits. Each constant shall represent a byte value. Multi-byte values can be represented by concatenated constants specified in byte order with the last constant specifying the least significant byte of the character.

   For example:

   ```
   \x63;\xe7;\x63\x68   "\x4d\x61\x79"
   ```

5. A character can be represented as a decimal constant. A decimal constant shall be specified as the escape character followed by a 'd' followed by two or three decimal digits. Each constant represents a byte value. Multi-byte values can be represented by concatenated constants specified in byte order with the last constant specifying the least significant byte of the character.

   For example:

   ```
   \d99;\d231;\d99\d104  "\d77\d97\d121"
   ```

Implementations may accept single-digit octal, decimal, or hexadecimal constants following the

escape character. Only characters existing in the character set for which the locale definition is created shall be specified, whether using symbolic names, the characters themselves, or octal, decimal, or hexadecimal constants. If a charmap file is present, only characters defined in the charmap can be specified using octal, decimal, or hexadecimal constants. Symbolic names not present in the charmap file can be specified and shall be ignored, as specified under item 1 above.

7.3.1 LC_CTYPE

The *LC_CTYPE* category shall define character classification, case conversion, and other character attributes. In addition, a series of characters can be represented by three adjacent <period> characters representing an ellipsis symbol (`"..."`). The ellipsis specification shall be interpreted as meaning that all values between the values preceding and following it represent valid characters. The ellipsis specification shall be valid only within a single encoded character set; that is, within a group of characters of the same size. An ellipsis shall be interpreted as including in the list all characters with an encoded value higher than the encoded value of the character preceding the ellipsis and lower than the encoded value of the character following the ellipsis.

For example:

```
\x30;...;\x39;
```

includes in the character class all characters with encoded values between the endpoints.

The following keywords shall be recognized. In the descriptions, the term ``automatically included'' means that it shall not be an error either to include or omit any of the referenced characters; the implementation provides them if missing (even if the entire keyword is missing) and accepts them silently if present. When the implementation automatically includes a missing character, it shall have an encoded value dependent on the charmap file in effect (see the description of the *localedef* **−f** option); otherwise, it shall have a value derived from an implementation-defined character mapping.

The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included characters. It is not possible to define a locale without these automatically included characters unless some implementation extension is used to prevent their inclusion. Such a definition would not be a proper superset of the C or POSIX locale and, thus, it might not be possible for conforming applications to work properly.

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

upper Define characters to be classified as uppercase letters.

In the POSIX locale, only:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

shall be included:

In a locale definition file, no character specified for the keywords **cntrl**, **digit**, **punct**, or **space** shall be specified. The uppercase letters <A> to <Z>, as defined in Section 6.4 (on page 121) (the portable character set), are automatically included in this class.

lower Define characters to be classified as lowercase letters.

In the POSIX locale, only:

4264 `a b c d e f g h i j k l m n o p q r s t u v w x y z`

4265 shall be included.

4266 In a locale definition file, no character specified for the keywords **cntrl**, **digit**,
4267 **punct**, or **space** shall be specified. The lowercase letters <a> to <z> of the
4268 portable character set are automatically included in this class.

4269 **alpha** Define characters to be classified as letters.

4270 In the POSIX locale, only characters in the classes **upper** and **lower** shall be
4271 included.

4272 In a locale definition file, no character specified for the keywords **cntrl**, **digit**,
4273 **punct**, or **space** shall be specified. Characters classified as either **upper** or
4274 **lower** are automatically included in this class.

4275 **digit** Define the characters to be classified as numeric digits.

4276 In all locales, only:

4277 `0 1 2 3 4 5 6 7 8 9`

4278 shall be included.

4279 In a locale definition file, only the digits <zero>, <one>, <two>, <three>,
4280 <four>, <five>, <six>, <seven>, <eight>, and <nine> shall be specified, and in
4281 contiguous ascending sequence by numerical value. The digits <zero> to
4282 <nine> of the portable character set are automatically included in this class.

4283 **alnum** Define characters to be classified as letters and numeric digits. Only the
4284 characters specified for the **alpha** and **digit** keywords shall be specified.
4285 Characters specified for the keywords **alpha** and **digit** are automatically
4286 included in this class.

4287 **space** Define characters to be classified as white-space characters.

4288 In the POSIX locale, exactly <space>, <form-feed>, <newline>, <carriage-
4289 return>, <tab>, and <vertical-tab> shall be included.

4290 In a locale definition file, no character specified for the keywords **upper**,
4291 **lower**, **alpha**, **digit**, **graph**, or **xdigit** shall be specified. The <space>, <form-
4292 feed>, <newline>, <carriage-return>, <tab>, and <vertical-tab> of the portable
4293 character set, and any characters included in the class **blank** are automatically
4294 included in this class.

4295 **cntrl** Define characters to be classified as control characters.

4296 In the POSIX locale, no characters in classes **alpha** or **print** shall be included.

4297 In a locale definition file, no character specified for the keywords **upper**,
4298 **lower**, **alpha**, **digit**, **punct**, **graph**, **print**, or **xdigit** shall be specified.

4299 **punct** Define characters to be classified as punctuation characters.

4300 In the POSIX locale, neither the <space> nor any characters in classes **alpha**,
4301 **digit**, or **cntrl** shall be included.

4302 In a locale definition file, no character specified for the keywords **upper**,
4303 **lower**, **alpha**, **digit**, **cntrl**, **xdigit**, or as the <space> shall be specified.

graph Define characters to be classified as printable characters, not including the <space>.

In the POSIX locale, all characters in classes **alpha**, **digit**, and **punct** shall be included; no characters in class **cntrl** shall be included.

In a locale definition file, characters specified for the keywords **upper**, **lower**, **alpha**, **digit**, **xdigit**, and **punct** are automatically included in this class. No character specified for the keyword **cntrl** shall be specified.

print Define characters to be classified as printable characters, including the <space>.

In the POSIX locale, all characters in class **graph** shall be included; no characters in class **cntrl** shall be included.

In a locale definition file, characters specified for the keywords **upper**, **lower**, **alpha**, **digit**, **xdigit**, **punct, graph**, and the <space> are automatically included in this class. No character specified for the keyword **cntrl** shall be specified.

xdigit Define the characters to be classified as hexadecimal digits.

In all locales, only:

```
0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f
```

shall be included.

In a locale definition file, only the characters defined for the class **digit** shall be specified, in contiguous ascending sequence by numerical value, followed by two sets, in either order, of six characters representing the hexadecimal digits corresponding to the decimal numbers 10 to 15 inclusive, with each set in ascending order: <A>, <B>, <C>, <D>, <E>, <F> and <a>, <b>, <c>, <d>, <e>, <f>. The digits <zero> to <nine>, the uppercase letters <A> to <F>, and the lowercase letters <a> to <f> of the portable character set are automatically included in this class.

blank Define characters to be classified as <blank> characters.

In the POSIX locale, only the <space> and <tab> shall be included.

In a locale definition file, no character specified for the keywords **upper**, **lower**, **alpha**, **digit**, **graph**, or **xdigit** shall be specified, and none of the characters <form-feed>, <newline>, <carriage-return>, and <vertical-tab> of the portable character set shall be specified. The <space> and <tab> are automatically included in this class.

charclass Define one or more locale-specific character class names as strings separated by <semicolon> characters. Each named character class can then be defined subsequently in the *LC_CTYPE* definition. A character class name shall consist of at least one and at most {CHARCLASS_NAME_MAX} bytes of alphanumeric characters from the portable filename character set. The first character of a character class name shall not be a digit. The name shall not match any of the *LC_CTYPE* keywords defined in this volume of POSIX.1-2024. Future versions of this standard will not specify any *LC_CTYPE* keywords containing uppercase letters.

charclass-name Define characters to be classified as belonging to the named locale-specific character class. In the POSIX locale, locale-specific named character classes need not exist.

If a class name is defined by a **charclass** keyword, but no characters are subsequently assigned to it, this is not an error; it represents a class without any characters belonging to it.

The *charclass-name* can be used as the *property* argument to the *wctype*() function, in regular expression and shell pattern-matching bracket expressions, and by the *tr* command.

toupper Define the mapping of lowercase letters to uppercase letters.

In the POSIX locale, the 26 lowercase characters:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

shall be mapped to the corresponding 26 uppercase characters:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

In a locale definition file, the operand shall consist of character pairs, separated by <semicolon> characters. The characters in each character pair shall be separated by a <comma> and the pair enclosed by parentheses. The first character in each pair is the lowercase letter, the second the corresponding uppercase letter. Only characters specified for the keywords **lower** and **upper** shall be specified. The lowercase letters <a> to <z>, and their corresponding uppercase letters <A> to <Z>, of the portable character set are automatically included in this mapping, but only when the **toupper** keyword is omitted from the locale definition.

tolower Define the mapping of uppercase letters to lowercase letters.

In the POSIX locale, the 26 uppercase characters:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

shall be mapped to the corresponding 26 lowercase characters:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

In a locale definition file, the operand shall consist of character pairs, separated by <semicolon> characters. The characters in each character pair shall be separated by a <comma> and the pair enclosed by parentheses. The first character in each pair is the uppercase letter, the second the corresponding lowercase letter. Only characters specified for the keywords **lower** and **upper** shall be specified. If the **tolower** keyword is omitted from the locale definition, the mapping is the reverse mapping of the one specified for **toupper**.

The following table shows the character class combinations allowed:

Table 7-1 Valid Character Class Combinations

	Can Also Belong To										
In Class	**upper**	**lower**	**alpha**	**digit**	**space**	**cntrl**	**punct**	**graph**	**print**	**xdigit**	**blank**
upper		—	A	x	x	x	x	A	A	—	x
lower	—		A	x	x	x	x	A	A	—	x
alpha	—	—		x	x	x	x	A	A	—	x
digit	x	x	x		x	x	x	A	A	A	x
space	x	x	x	x		—	*	*	*	x	—
cntrl	x	x	x	x	—		x	x	x	x	—
punct	x	x	x	x	—	x		A	A	x	—
graph	—	—	—	—	—	x	—		A	—	—
print	—	—	—	—	—	x	—	—		—	—
xdigit	—	—	—	—	x	x	x	A	A		x
blank	x	x	x	x	A	—	*	*	*	x	

Notes:

1. Explanation of codes:

   A Automatically included; see text.

   — Permitted.

   x Mutually-exclusive.

   * See note 2.

2. The <space>, which is part of the **space** and **blank** classes, cannot belong to **punct** or **graph**, but shall automatically belong to the **print** class. Other **space** or **blank** characters can be classified as any of **punct**, **graph**, or **print**.

7.3.1.1 *LC_CTYPE Category in the POSIX Locale*

The minimum character classifications for the POSIX locale follow; the code listing depicts the *localedef* input, and the table represents the same information, sorted by character. Implementations may add additional characters to the **cntrl** and **punct** classifications but shall not make any other additions.

```
LC_CTYPE
# The following is the minimum POSIX locale LC_CTYPE.
# "alpha" is by definition "upper" and "lower"
# "alnum" is by definition "alpha" and "digit"
# "print" is by definition "alnum", "punct", and the <space>
# "graph" is by definition "alnum" and "punct"
#
upper    <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;<M>;\
         <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>
#
lower    <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
         <n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>
#
digit    <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
         <seven>;<eight>;<nine>
#
space    <tab>;<newline>;<vertical-tab>;<form-feed>;\
```



```
4428                <carriage-return>;<space>
4429        #
4430        cntrl   <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\
4431                <form-feed>;<carriage-return>;\
4432                <NUL>;<SOH>;<STX>;<ETX>;<EOT>;<ENQ>;<ACK>;<SO>;\
4433                <SI>;<DLE>;<DC1>;<DC2>;<DC3>;<DC4>;<NAK>;<SYN>;\
4434                <ETB>;<CAN>;<EM>;<SUB>;<ESC>;<IS4>;<IS3>;<IS2>;\
4435                <IS1>;<DEL>
4436        #
4437        punct   <exclamation-mark>;<quotation-mark>;<number-sign>;\
4438                <dollar-sign>;<percent-sign>;<ampersand>;<apostrophe>;\
4439                <left-parenthesis>;<right-parenthesis>;<asterisk>;\
4440                <plus-sign>;<comma>;<hyphen-minus>;<period>;<slash>;\
4441                <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
4442                <greater-than-sign>;<question-mark>;<commercial-at>;\
4443                <left-square-bracket>;<backslash>;<right-square-bracket>;\
4444                <circumflex>;<underscore>;<grave-accent>;<left-curly-bracket>;\
4445                <vertical-line>;<right-curly-bracket>;<tilde>
4446        #
4447        xdigit  <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;\
4448                <eight>;<nine>;<A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
4449        #
4450        blank   <space>;<tab>
4451        #
4452        toupper (<a>,<A>);(<b>,<B>);(<c>,<C>);(<d>,<D>);(<e>,<E>);\
4453                (<f>,<F>);(<g>,<G>);(<h>,<H>);(<i>,<I>);(<j>,<J>);\
4454                (<k>,<K>);(<l>,<L>);(<m>,<M>);(<n>,<N>);(<o>,<O>);\
4455                (<p>,<P>);(<q>,<Q>);(<r>,<R>);(<s>,<S>);(<t>,<T>);\
4456                (<u>,<U>);(<v>,<V>);(<w>,<W>);(<x>,<X>);(<y>,<Y>);(<z>,<Z>)
4457        #
4458        tolower (<A>,<a>);(<B>,<b>);(<C>,<c>);(<D>,<d>);(<E>,<e>);\
4459                (<F>,<f>);(<G>,<g>);(<H>,<h>);(<I>,<i>);(<J>,<j>);\
4460                (<K>,<k>);(<L>,<l>);(<M>,<m>);(<N>,<n>);(<O>,<o>);\
4461                (<P>,<p>);(<Q>,<q>);(<R>,<r>);(<S>,<s>);(<T>,<t>);\
4462                (<U>,<u>);(<V>,<v>);(<W>,<w>);(<X>,<x>);(<Y>,<y>);(<Z>,<z>)
4463        END LC_CTYPE
```


Symbolic Name	Other Case	Character Classes
<NUL>		**cntrl**
<SOH>		**cntrl**
<STX>		**cntrl**
<ETX>		**cntrl**
<EOT>		**cntrl**
<ENQ>		**cntrl**
<ACK>		**cntrl**
<alert>		**cntrl**
<backspace>		**cntrl**
<tab>		**cntrl, space, blank**
<newline>		**cntrl, space**
<vertical-tab>		**cntrl, space**
<form-feed>		**cntrl, space**
<carriage-return>		**cntrl, space**
<SO>		**cntrl**
<SI>		**cntrl**
<DLE>		**cntrl**
<DC1>		**cntrl**
<DC2>		**cntrl**
<DC3>		**cntrl**
<DC4>		**cntrl**
<NAK>		**cntrl**
<SYN>		**cntrl**
<ETB>		**cntrl**
<CAN>		**cntrl**
<EM>		**cntrl**
<SUB>		**cntrl**
<ESC>		**cntrl**
<IS4>		**cntrl**
<IS3>		**cntrl**
<IS2>		**cntrl**
<IS1>		**cntrl**
<space>		**space, print, blank**
<exclamation-mark>		**punct, print, graph**
<quotation-mark>		**punct, print, graph**
<number-sign>		**punct, print, graph**
<dollar-sign>		**punct, print, graph**
<percent-sign>		**punct, print, graph**
<ampersand>		**punct, print, graph**
<apostrophe>		**punct, print, graph**
<left-parenthesis>		**punct, print, graph**
<right-parenthesis>		**punct, print, graph**
<asterisk>		**punct, print, graph**
<plus-sign>		**punct, print, graph**
<comma>		**punct, print, graph**
<hyphen-minus>		**punct, print, graph**
<period>		**punct, print, graph**
<slash>		**punct, print, graph**
<zero>		**digit, xdigit, print, graph**
<one>		**digit, xdigit, print, graph**
<two>		**digit, xdigit, print, graph**

Symbolic Name	Other Case	Character Classes
<three>		**digit, xdigit, print, graph**
<four>		**digit, xdigit, print, graph**
<five>		**digit, xdigit, print, graph**
<six>		**digit, xdigit, print, graph**
<seven>		**digit, xdigit, print, graph**
<eight>		**digit, xdigit, print, graph**
<nine>		**digit, xdigit, print, graph**
<colon>		**punct, print, graph**
<semicolon>		**punct, print, graph**
<less-than-sign>		**punct, print, graph**
<equals-sign>		**punct, print, graph**
<greater-than-sign>		**punct, print, graph**
<question-mark>		**punct, print, graph**
<commercial-at>		**punct, print, graph**
<A>	<a>	**upper, xdigit, alpha, print, graph**
<B>	<b>	**upper, xdigit, alpha, print, graph**
<C>	<c>	**upper, xdigit, alpha, print, graph**
<D>	<d>	**upper, xdigit, alpha, print, graph**
<E>	<e>	**upper, xdigit, alpha, print, graph**
<F>	<f>	**upper, xdigit, alpha, print, graph**
<G>	<g>	**upper, alpha, print, graph**
<H>	<h>	**upper, alpha, print, graph**
<I>	<i>	**upper, alpha, print, graph**
<J>	<j>	**upper, alpha, print, graph**
<K>	<k>	**upper, alpha, print, graph**
<L>	<l>	**upper, alpha, print, graph**
<M>	<m>	**upper, alpha, print, graph**
<N>	<n>	**upper, alpha, print, graph**
<O>	<o>	**upper, alpha, print, graph**
<P>	<p>	**upper, alpha, print, graph**
<Q>	<q>	**upper, alpha, print, graph**
<R>	<r>	**upper, alpha, print, graph**
<S>	<s>	**upper, alpha, print, graph**
<T>	<t>	**upper, alpha, print, graph**
<U>	<u>	**upper, alpha, print, graph**
<V>	<v>	**upper, alpha, print, graph**
<W>	<w>	**upper, alpha, print, graph**
<X>	<x>	**upper, alpha, print, graph**
<Y>	<y>	**upper, alpha, print, graph**
<Z>	<z>	**upper, alpha, print, graph**
<left-square-bracket>		**punct, print, graph**
<backslash>		**punct, print, graph**
<right-square-bracket>		**punct, print, graph**
<circumflex>		**punct, print, graph**
<underscore>		**punct, print, graph**
<grave-accent>		**punct, print, graph**
<a>	<A>	**lower, xdigit, alpha, print, graph**
<b>	<B>	**lower, xdigit, alpha, print, graph**
<c>	<C>	**lower, xdigit, alpha, print, graph**
<d>	<D>	**lower, xdigit, alpha, print, graph**
<e>	<E>	**lower, xdigit, alpha, print, graph**

Symbolic Name	Other Case	Character Classes
<f>	<F>	**lower, xdigit, alpha, print, graph**
<g>	<G>	**lower, alpha, print, graph**
<h>	<H>	**lower, alpha, print, graph**
<i>	<I>	**lower, alpha, print, graph**
<j>	<J>	**lower, alpha, print, graph**
<k>	<K>	**lower, alpha, print, graph**
<l>	<L>	**lower, alpha, print, graph**
<m>	<M>	**lower, alpha, print, graph**
<n>	<N>	**lower, alpha, print, graph**
<o>	<O>	**lower, alpha, print, graph**
<p>	<P>	**lower, alpha, print, graph**
<q>	<Q>	**lower, alpha, print, graph**
<r>	<R>	**lower, alpha, print, graph**
<s>	<S>	**lower, alpha, print, graph**
<t>	<T>	**lower, alpha, print, graph**
<u>	<U>	**lower, alpha, print, graph**
<v>	<V>	**lower, alpha, print, graph**
<w>	<W>	**lower, alpha, print, graph**
<x>	<X>	**lower, alpha, print, graph**
<y>	<Y>	**lower, alpha, print, graph**
<z>	<Z>	**lower, alpha, print, graph**
<left-curly-bracket>		**punct, print, graph**
<vertical-line>		**punct, print, graph**
<right-curly-bracket>		**punct, print, graph**
<tilde>		**punct, print, graph**
<DEL>		**cntrl**

7.3.2 LC_COLLATE

The *LC_COLLATE* category provides a collation sequence definition for numerous utilities in the Shell and Utilities volume of POSIX.1-2024 (*ls*, *sort*, and so on), regular expression matching (see Chapter 9, on page 179), and the *strcoll*( ), *strxfrm*( ), *wcscoll*( ), and *wcsxfrm*( ) functions in the System Interfaces volume of POSIX.1-2024.

A collation sequence definition shall define the relative order between collating elements (characters and multi-character collating elements) in the locale. This order is expressed in terms of collation values; that is, by assigning each element one or more collation values (also known as collation weights). This does not imply that implementations shall assign such values, but that ordering of strings using the resultant collation definition in the locale behaves as if such assignment is done and used in the collation process. At least the following capabilities are provided:

1. **Multi-character collating elements**. Specification of multi-character collating elements (that is, sequences of two or more characters to be collated as an entity).
2. **User-defined ordering of collating elements**. Each collating element shall be assigned a collation value defining its order in the character (or basic) collation sequence. This ordering is used by regular expressions and pattern matching and, unless collation weights are explicitly specified, also as the collation weight to be used in sorting.
3. **Multiple weights and equivalence classes**. Collating elements can be assigned one or more (up to the limit {COLL_WEIGHTS_MAX}, as defined in **<limits.h>**) collating weights for use in sorting. The first weight is hereafter referred to as the primary weight.

4. **One-to-many mapping**. A single character is mapped into a string of collating elements.
5. **Equivalence class definition**. Two or more collating elements have the same collation value (primary weight).
6. **Ordering by weights**. When two strings are compared to determine their relative order, the two strings are first broken up into a series of collating elements; the elements in each successive pair of elements are then compared according to the relative primary weights for the elements. If equal, and more than one weight has been assigned, then the pairs of collating elements are re-compared according to the relative subsequent weights, until either a pair of collating elements compare unequal or the weights are exhausted.

All implementation-provided locales (either preinstalled or provided as locale definitions which can be installed later) shall define a collation sequence that has a total ordering of all characters unless the locale name has an `'@'` modifier indicating that it has a special collation sequence (for example, `@icase` could indicate that each upper and lowercase character pair collates equally).

Note: Users installing their own locales should ensure that they define a collation sequence with a total ordering of all characters unless an `'@'` modifier in the locale name (such as `@icase`) indicates that it has a special collation sequence. As <NUL> is reserved as the string terminator for most usages of *LC_COLLATE*, it is the responsibility of the locale writer to ensure <NUL> has the lowest primary weight in a collation ordering for the interfaces to behave in the way users typically expect. Unusual behavior may result if it has any other collation order weighting, or is subject to **IGNORE**.

The following keywords shall be recognized in a collation sequence definition. They are described in detail in the following sections.

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

collating-element Define a collating-element symbol representing a multi-character collating element. This keyword is optional.

collating-symbol Define a collating symbol for use in collation order statements. This keyword is optional.

order_start Define collation rules. This statement shall be followed by one or more collation order statements, assigning character collation values and collation weights to collating elements.

order_end Specify the end of the collation-order statements.

7.3.2.1 *The collating-element Keyword*

In addition to the collating elements in the character set, the **collating-element** keyword can be used to define multi-character collating elements. The syntax is as follows:

```
"collating-element %s from \"%s\"\n", <collating-symbol>, <string>
```

The <*collating-symbol*> operand shall be a symbolic name, enclosed between angle brackets (`'<'` and `'>'`), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. The string operand is a string of two or more characters that collates as an entity. A <*collating-element*> defined via this keyword is only recognized with the *LC_COLLATE* category.

For example:

```
collating-element <ch> from "<c><h>"
```



```
collating-element <e-acute> from "<acute><e>"
collating-element <ll> from "ll"
```

7.3.2.2 The collating-symbol Keyword

This keyword shall be used to define symbols for use in collation sequence statements; that is, between the **order_start** and the **order_end** keywords. The syntax is as follows:

```
"collating-symbol %s\n", <collating-symbol>
```

The <*collating-symbol*> shall be a symbolic name, enclosed between angle brackets ('<' and '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. A <*collating-symbol*> defined via this keyword is only recognized within the *LC_COLLATE* category.

For example:

```
collating-symbol <UPPER_CASE>
collating-symbol <HIGH>
```

The **collating-symbol** keyword defines a symbolic name that can be associated with a relative position in the character order sequence. While such a symbolic name does not represent any collating element, it can be used as a weight.

7.3.2.3 The order_start Keyword

The **order_start** keyword shall precede collation order entries and also define the number of weights for this collation sequence definition and other collation rules. The syntax is as follows:

```
"order_start %s;%s;...;%s\n", <sort-rules>, <sort-rules> ...
```

The operands to the **order_start** keyword are optional. If present, the operands define rules to be applied when strings are compared. The number of operands define how many weights each element is assigned; if no operands are present, one **forward** operand is assumed. If present, the first operand defines rules to be applied when comparing strings using the first (primary) weight; the second when comparing strings using the second weight, and so on. Operands shall be separated by <semicolon> characters (';'). Each operand shall consist of one or more collation directives, separated by <comma> characters (','). If the number of operands exceeds the {COLL_WEIGHTS_MAX} limit, the utility shall issue a warning message. The following directives shall be supported:

forward Specifies that comparison operations for the weight level shall proceed from start of string towards the end of string.

backward Specifies that comparison operations for the weight level shall proceed from end of string towards the beginning of string.

position Specifies that comparison operations for the weight level shall consider the relative position of elements in the strings not subject to **IGNORE**. The string containing an element not subject to **IGNORE** after the fewest collating elements subject to **IGNORE** from the start of the compare shall collate first. If both strings contain a character not subject to **IGNORE** in the same relative position, the collating values assigned to the elements shall determine the ordering. In case of equality, subsequent characters not subject to **IGNORE** shall be considered in the same manner.

The directives **forward** and **backward** are mutually-exclusive.

If no operands are specified, a single **forward** operand shall be assumed.

For example:

```
order_start     forward;backward
```

7.3.2.4 *Collation Order*

The **order_start** keyword shall be followed by collating identifier entries. The syntax for the collating element entries is as follows:

```
"%s %s;%s;...;%s\n", <collating-identifier>, <weight>, <weight>, ...
```

Each *collating-identifier* shall consist of either a character (in any of the forms defined in Section 7.3, on page 128), a <*collating-element*>, a <*collating-symbol*>, an ellipsis, or the special symbol **UNDEFINED**. The order in which collating elements are specified determines the character order sequence, such that each collating element shall compare less than the elements following it.

A <*collating-element*> shall be used to specify multi-character collating elements, and indicates that the character sequence specified via the <*collating-element*> is to be collated as a unit and in the relative order specified by its place.

A <*collating-symbol*> can be used to define a position in the relative order for use in weights. No weights shall be specified with a <*collating-symbol*>.

The ellipsis symbol specifies that a sequence of characters shall collate according to their encoded character values. It shall be interpreted as indicating that all characters with a coded character set value higher than the value of the character in the preceding line, and lower than the coded character set value for the character in the following line, in the current coded character set, shall be placed in the character collation order between the previous and the following character in ascending order according to their coded character set values. An initial ellipsis shall be interpreted as if the preceding line specified the NUL character, and a trailing ellipsis as if the following line specified the highest coded character set value in the current coded character set. An ellipsis shall be treated as invalid if the preceding or following lines do not specify characters in the current coded character set. The use of the ellipsis symbol ties the definition to a specific coded character set and may preclude the definition from being portable between implementations.

The symbol **UNDEFINED** shall be interpreted as including all coded character set values not specified explicitly or via the ellipsis symbol. Such characters shall be inserted in the character collation order at the point indicated by the symbol, and in ascending order according to their coded character set values. If no **UNDEFINED** symbol is specified, and the current coded character set contains characters not specified in this section, the utility shall issue a warning message and place such characters at the end of the character collation order.

The optional operands for each collation-element shall be used to define the primary, secondary, or subsequent weights for the collating element. The first operand specifies the relative primary weight, the second the relative secondary weight, and so on. Two or more collation-elements can be assigned the same weight; they belong to the same ``equivalence class'' if they have the same primary weight. Collation shall behave as if, for each weight level, elements subject to **IGNORE** are removed, unless the **position** collation directive is specified for the corresponding level with the **order_start** keyword. Then each successive pair of elements shall be compared according to the relative weights for the elements. If the two strings compare equal, the process shall be repeated for the next weight level, up to the limit {COLL_WEIGHTS_MAX}.

Weights shall be assigned such that the collation sequence has a total ordering of all characters

unless an `'@'` modifier in the locale name indicates that it has a special collation sequence.

Weights shall be expressed as characters (in any of the forms specified in Section 7.3, on page 128), <*collating-symbol*>s, <*collating-element*>s, an ellipsis, or the special symbol **IGNORE**. A single character, a <*collating-symbol*>, or a <*collating-element*> shall represent the relative position in the character collating sequence of the character or symbol, rather than the character or characters themselves. Thus, rather than assigning absolute values to weights, a particular weight is expressed using the relative order value assigned to a collating element based on its order in the character collation sequence.

One-to-many mapping is indicated by specifying two or more concatenated characters or symbolic names. For example, if the <eszet> is given the string `"<s><s>"` as a weight, comparisons are performed as if all occurrences of the <eszet> are replaced by `"<s><s>"` (assuming that `"<s>"` has the collating weight `"<s>"`). If it is necessary to define <eszet> and `"<s><s>"` as an equivalence class, then a collating element needs to be defined for the string `"ss"`.

All characters specified via an ellipsis shall by default be assigned unique weights, equal to the relative order of characters. Characters specified via an explicit or implicit **UNDEFINED** special symbol shall by default be assigned the same primary weight (that is, they belong to the same equivalence class) if the collation order has more than one weight level. If the collation order has only one weight level, these characters shall be assigned unique primary weights, equal to the relative order of their character in the character collation sequence.

An ellipsis symbol as a weight shall be interpreted to mean that each character in the sequence shall have unique weights, equal to the relative order of their character in the character collation sequence. The use of the ellipsis as a weight shall be treated as an error if the collating element is neither an ellipsis nor the special symbol **UNDEFINED**.

The special keyword **IGNORE** as a weight shall indicate that when strings are compared using the weights at the level where **IGNORE** is specified, the collating element shall be ignored; that is, as if the string did not contain the collating element. In regular expressions and pattern matching, all characters that are subject to **IGNORE** in their primary weight form an equivalence class.

An empty operand shall be interpreted as the collating element itself.

For example, the order statement:

```
<a>     <a>;<a>
```

is equal to:

```
<a>
```

An ellipsis can be used as an operand if the collating element was an ellipsis, and shall be interpreted as the value of each character defined by the ellipsis.

The collation order as defined in this section affects the interpretation of bracket expressions in regular expressions (see Section 9.3.5, on page 182).

For example:

```
order_start  forward;backward
<NUL>        <NUL>;<NUL>
<LOW>
<space>      <LOW>;<space>
...          <LOW>;...
<a>          <a>;<a>
<a-acute>    <a>;<a-acute>
<a-grave>    <a>;<a-grave>
<A>          <a>;<A>
<A-acute>    <a>;<A-acute>
<A-grave>    <a>;<A-grave>
<ch>         <ch>;<ch>
<Ch>         <ch>;<Ch>
<s>          <s>;<s>
<eszet>      "<s><s>";"<eszet><eszet>"
UNDEFINED    IGNORE;...
order_end
```

This example is interpreted as follows:

1. All characters between <space> and `'a'` shall have the same primary equivalence class and individual secondary weights based on their ordinal encoded values.
2. All characters based on the uppercase or lowercase character `'a'` belong to the same primary equivalence class.
3. The multi-character collating element <ch> is represented by the collating symbol <ch> and belongs to the same primary equivalence class as the multi-character collating element <Ch>.
4. The **UNDEFINED** means that all characters not specified in this definition (explicitly or via the ellipsis) shall be ignored when comparing primary weights, and have individual secondary weights based on their ordinal encoded values.

7.3.2.5 *The order_end Keyword*

The collating order entries shall be terminated with an **order_end** keyword.

7.3.2.6 *LC_COLLATE Category in the POSIX Locale*

The minimum collation sequence definition of the POSIX locale follows; the code listing depicts the *localedef* input. All characters not explicitly listed here shall be inserted in the character collation order after the listed characters and shall be assigned unique primary weights. If the listed characters have ASCII encoding, the other characters shall be in ascending order according to their coded character set values; otherwise, the order of the other characters is unspecified. The collation sequence shall not include any multi-character collating elements.

```
LC_COLLATE
# This is the minimum input for the POSIX locale definition for the
# LC_COLLATE category. Characters in this list are in the same order
# as in the ASCII codeset.
order_start forward
<NUL>
<SOH>
```



```
        <STX>
        <ETX>
        <EOT>
        <ENQ>
        <ACK>
        <alert>
        <backspace>
        <tab>
        <newline>
        <vertical-tab>
        <form-feed>
        <carriage-return>
        <SO>
        <SI>
        <DLE>
        <DC1>
        <DC2>
        <DC3>
        <DC4>
        <NAK>
        <SYN>
        <ETB>
        <CAN>
        <EM>
        <SUB>
        <ESC>
        <IS4>
        <IS3>
        <IS2>
        <IS1>
        <space>
        <exclamation-mark>
        <quotation-mark>
        <number-sign>
        <dollar-sign>
        <percent-sign>
        <ampersand>
        <apostrophe>
        <left-parenthesis>
        <right-parenthesis>
        <asterisk>
        <plus-sign>
        <comma>
        <hyphen-minus>
        <period>
        <slash>
        <zero>
        <one>
        <two>
        <three>
        <four>
        <five>
        <six>
```



```
        <seven>
        <eight>
        <nine>
        <colon>
        <semicolon>
        <less-than-sign>
        <equals-sign>
        <greater-than-sign>
        <question-mark>
        <commercial-at>
        <A>
        <B>
        <C>
        <D>
        <E>
        <F>
        <G>
        <H>
        <I>
        <J>
        <K>
        <L>
        <M>
        <N>
        <O>
        <P>
        <Q>
        <R>
        <S>
        <T>
        <U>
        <V>
        <W>
        <X>
        <Y>
        <Z>
        <left-square-bracket>
        <backslash>
        <right-square-bracket>
        <circumflex>
        <underscore>
        <grave-accent>
        <a>
        <b>
        <c>
        <d>
        <e>
        <f>
        <g>
        <h>
        <i>
        <j>
        <k>
```



```
<l>
<m>
<n>
<o>
<p>
<q>
<r>
<s>
<t>
<u>
<v>
<w>
<x>
<y>
<z>
<left-curly-bracket>
<vertical-line>
<right-curly-bracket>
<tilde>
<DEL>
order_end
#
END LC_COLLATE
```

7.3.3 LC_MONETARY

The *LC_MONETARY* category shall define the rules and symbols that are used to format monetary numeric information.

This information is available through the *localeconv*() function and is used by the *strfmon*() function.

Some of the information is also available in an alternative form via the *nl_langinfo*() function (see CRNCYSTR in **<langinfo.h>**).

The following items are defined in this category of the locale. The item names are the keywords recognized by the *localedef* utility when defining a locale. They are also similar to the member names of the **lconv** structure defined in **<locale.h>**; see **<locale.h>** for the exact symbols in the header. The *localeconv*() function returns {CHAR_MAX} for unavailable integer items and the empty string (`""`) for unavailable or size zero string items.

In a locale definition file, the operands are strings, formatted as indicated by the grammar in Section 7.4 (on page 160). For some keywords, the strings can contain only integers. Keywords that are not provided, or integer keywords set to –1, can be used to indicate that the value is not available in the locale. String values set to the empty string (`""`) can be used to indicate that the value is available and is an empty string, or that the value is not available. The following keywords shall be recognized:

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

Note: This is a *localedef* utility keyword, unavailable through *localeconv*().

int_curr_symbol The international currency symbol. The operand shall be a four-character string, with the first three characters containing the alphabetic international currency symbol. The international currency symbol should be chosen in accordance with those specified in the ISO 4217 standard. The fourth character shall be the character used to separate the international currency symbol from the monetary quantity.

currency_symbol The string that shall be used as the local currency symbol.

mon_decimal_point The operand is a string containing the symbol that shall be used as the decimal delimiter (radix character) in monetary formatted quantities.

mon_thousands_sep The operand is a string containing the symbol that shall be used as a separator for groups of digits to the left of the decimal delimiter in formatted monetary quantities.

mon_grouping Define the size of each group of digits in formatted monetary quantities. The operand is a sequence of integers separated by <semicolon> characters. Each integer specifies the number of digits in each group, with the initial integer defining the size of the group immediately preceding the decimal delimiter, and the following integers defining the preceding groups. If the last integer is not –1, then the size of the previous group (if any) shall be repeatedly used for the remainder of the digits. If the last integer is –1, then no further grouping shall be performed.

positive_sign A string that shall be used to indicate a non-negative-valued formatted monetary quantity.

negative_sign A string that shall be used to indicate a negative-valued formatted monetary quantity.

int_frac_digits An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using **int_curr_symbol**.

frac_digits An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using **currency_symbol**.

p_cs_precedes An integer set to 1 if the **currency_symbol** precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.

p_sep_by_space Set to a value indicating the separation of the **currency_symbol**, the sign string, and the value for a non-negative formatted monetary quantity.

The values of **p_sep_by_space**, **n_sep_by_space**, **int_p_sep_by_space**, and **int_n_sep_by_space** are interpreted according to the following:

0 No <space> separates the currency symbol and value.

1 If the currency symbol and sign string are adjacent, a <space> separates them from the value; otherwise, a <space> separates the currency symbol from the value.

2 If the currency symbol and sign string are adjacent, a <space> separates them; otherwise, a <space> separates the sign string from the value.

n_cs_precedes An integer set to 1 if the **currency_symbol** precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.

n_sep_by_space Set to a value indicating the separation of the **currency_symbol**, the sign string, and the value for a negative formatted monetary quantity.

p_sign_posn An integer set to a value indicating the positioning of the **positive_sign** for a monetary quantity with a non-negative value. The following integer values shall be recognized for **int_n_sign_posn**, **int_p_sign_posn**, **n_sign_posn**, and **p_sign_posn**:

0 Parentheses enclose the quantity and the **currency_symbol**.

1 The sign string precedes the quantity and the **currency_symbol**.

2 The sign string succeeds the quantity and the **currency_symbol**.

3 The sign string precedes the **currency_symbol**.

4 The sign string succeeds the **currency_symbol**.

n_sign_posn An integer set to a value indicating the positioning of the **negative_sign** for a negative formatted monetary quantity.

int_p_cs_precedes An integer set to 1 if the **int_curr_symbol** precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.

int_n_cs_precedes An integer set to 1 if the **int_curr_symbol** precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.

int_p_sep_by_space Set to a value indicating the separation of the **int_curr_symbol**, the sign string, and the value for a non-negative internationally formatted monetary quantity.

int_n_sep_by_space Set to a value indicating the separation of the **int_curr_symbol**, the sign string, and the value for a negative internationally formatted monetary quantity.

int_p_sign_posn An integer set to a value indicating the positioning of the **positive_sign** for a positive monetary quantity formatted with the international format.

int_n_sign_posn An integer set to a value indicating the positioning of the **negative_sign** for a negative monetary quantity formatted with the international format.

Certain combinations of the ***_sign_posn**, **positive_sign**, and **negative_sign** values are invalid and shall not be accepted by *localedef*:

- If **p_sign_posn** and **n_sign_posn** are both greater than 0, and **positive_sign** and **negative_sign** have the same value or are both either empty strings or omitted, this combination is invalid because it requires signs to be used but does not provide the means to distinguish negative from positive values using signs.
- Likewise, if **int_p_sign_posn** and **int_n_sign_posn** are both greater than 0, and **positive_sign** and **negative_sign** have the same value or are both either empty strings or omitted, this combination is invalid.
- If **p_sign_posn** and **n_sign_posn** are both 0, this combination is invalid because it requires parentheses to be used but does not provide the means to distinguish negative from positive values using parentheses.

- Likewise, if **int_p_sign_posn** and **int_n_sign_posn** are both 0, this combination is invalid.

7.3.3.1 *LC_MONETARY Category in the POSIX Locale*

The monetary formatting definitions for the POSIX locale follow; the code listing depicting the *localedef* input, the table representing the same information with the addition of *localeconv*( ) and *nl_langinfo*( ) formats. All values shall be unavailable in the POSIX locale.

```
LC_MONETARY
# This is the POSIX locale definition for
# the LC_MONETARY category.
#
int_curr_symbol      ""
currency_symbol      ""
mon_decimal_point    ""
mon_thousands_sep    ""
mon_grouping         -1
positive_sign        ""
negative_sign        ""
int_frac_digits      -1
frac_digits          -1
p_cs_precedes        -1
p_sep_by_space       -1
n_cs_precedes        -1
n_sep_by_space       -1
p_sign_posn          -1
n_sign_posn          -1
int_p_cs_precedes    -1
int_p_sep_by_space   -1
int_n_cs_precedes    -1
int_n_sep_by_space   -1
int_p_sign_posn      -1
int_n_sign_posn      -1
#
END LC_MONETARY
```


Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value
int_curr_symbol	—	N/A	""	""
currency_symbol	CRNCYSTR	N/A	""	""
mon_decimal_point	—	N/A	""	""
mon_thousands_sep	—	N/A	""	""
mon_grouping	—	N/A	""	–1
positive_sign	—	N/A	""	""
negative_sign	—	N/A	""	""
int_frac_digits	—	N/A	{CHAR_MAX}	–1
frac_digits	—	N/A	{CHAR_MAX}	–1
p_cs_precedes	CRNCYSTR	N/A	{CHAR_MAX}	–1
p_sep_by_space	—	N/A	{CHAR_MAX}	–1
n_cs_precedes	CRNCYSTR	N/A	{CHAR_MAX}	–1
n_sep_by_space	—	N/A	{CHAR_MAX}	–1
p_sign_posn	—	N/A	{CHAR_MAX}	–1
n_sign_posn	—	N/A	{CHAR_MAX}	–1
int_p_cs_precedes	—	N/A	{CHAR_MAX}	–1
int_p_sep_by_space	—	N/A	{CHAR_MAX}	–1
int_n_cs_precedes	—	N/A	{CHAR_MAX}	–1
int_n_sep_by_space	—	N/A	{CHAR_MAX}	–1
int_p_sign_posn	—	N/A	{CHAR_MAX}	–1
int_n_sign_posn	—	N/A	{CHAR_MAX}	–1

The entry N/A indicates that the value is not available in the POSIX locale.

7.3.4 LC_NUMERIC

The *LC_NUMERIC* category shall define the rules and symbols that are used to format non-monetary numeric information. This information is available through the *localeconv*( ) function.

Some of the information is also available in an alternative form via the *nl_langinfo*( ) function.

The following items are defined in this category of the locale. The item names are the keywords recognized by the *localedef* utility when defining a locale. They are also similar to the member names of the **lconv** structure defined in **<locale.h>**; see **<locale.h>** for the exact symbols in the header. The *localeconv*( ) function returns {CHAR_MAX} for unavailable integer items and the empty string (**""**) for unavailable or size zero string items.

In a locale definition file, the operands are strings, formatted as indicated by the grammar in Section 7.4 (on page 160). For some keywords, the strings can only contain integers. Keywords that are not provided, or integer keywords set to –1, can be used to indicate that the value is not available in the locale. String values set to the empty string (**""**) can be used to indicate that the value is available and is an empty string, or that the value is not available. The following keywords shall be recognized:

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

Note: This is a *localedef* utility keyword, unavailable through *localeconv*( ).

decimal_point The operand is a string containing the symbol that shall be used as the decimal delimiter (radix character) in numeric, non-monetary formatted quantities. This keyword cannot be omitted and cannot be set to the empty string. In contexts where standards limit the **decimal_point** to a single byte,

the result of specifying a multi-byte operand shall be unspecified.

thousands_sep The operand is a string containing the symbol that shall be used as a separator for groups of digits to the left of the decimal delimiter in numeric, non-monetary formatted monetary quantities. In contexts where standards limit the **thousands_sep** to a single byte, the result of specifying a multi-byte operand shall be unspecified.

grouping Define the size of each group of digits in formatted non-monetary quantities. The operand is a sequence of integers separated by <semicolon> characters. Each integer specifies the number of digits in each group, with the initial integer defining the size of the group immediately preceding the decimal delimiter, and the following integers defining the preceding groups. If the last integer is not –1, then the size of the previous group (if any) shall be repeatedly used for the remainder of the digits. If the last integer is –1, then no further grouping shall be performed.

7.3.4.1 *LC_NUMERIC Category in the POSIX Locale*

The non-monetary numeric formatting definitions for the POSIX locale follow; the code listing depicting the *localedef* input, the table representing the same information with the addition of *localeconv*( ) values, and *nl_langinfo*( ) constants.

```
LC_NUMERIC
# This is the POSIX locale definition for
# the LC_NUMERIC category.
#
decimal_point    "<period>"
thousands_sep    ""
grouping         -1
#
END LC_NUMERIC
```

Item	langinfo Constant	POSIX Locale Value	localeconv( ) Value	localedef Value
decimal_point	RADIXCHAR	"."	"."	.
thousands_sep	THOUSEP	N/A	""	""
grouping	—	N/A	""	–1

The entry N/A indicates that the value is not available in the POSIX locale.

7.3.5 LC_TIME

The *LC_TIME* category shall define the interpretation of the conversion specifications supported by the *date* utility and shall affect the behavior of the *strftime*( ), *wcsftime*( ), *strptime*( ), and *nl_langinfo*( ) functions. Since the interfaces for C-language access and locale definition differ significantly, they are described separately.

5185 *7.3.5.1 LC_TIME Locale Definition*

5186 In a locale definition, the following mandatory keywords shall be recognized:

5187 **copy** Specify the name of an existing locale which shall be used as the definition of
5188 this category. If this keyword is specified, no other keyword shall be specified.

5189 **abday** Define the abbreviated weekday names, corresponding to the `%a` conversion
5190 specification (conversion specification in the *strftime*(), *wcsftime*(), and
5191 *strptime*() functions). The operand shall consist of seven
5192 <semicolon>-separated strings, each surrounded by double-quotes. The first
5193 string shall be the abbreviated name of the day corresponding to Sunday, the
5194 second the abbreviated name of the day corresponding to Monday, and so on.

5195 **day** Define the full weekday names, corresponding to the `%A` conversion
5196 specification. The operand shall consist of seven <semicolon>-separated
5197 strings, each surrounded by double-quotes. The first string is the full name of
5198 the day corresponding to Sunday, the second the full name of the day
5199 corresponding to Monday, and so on.

5200 **abmon** Define the abbreviated month names, corresponding to the `%b` conversion
5201 specification. The operand shall consist of twelve <semicolon>-separated
5202 strings, each surrounded by double-quotes. The first string shall be the
5203 abbreviated name of the first month of the year (January), the second the
5204 abbreviated name of the second month, and so on. For languages having both
5205 a genitive (when used with a day number) and a nominative (no day number)
5206 case, this operand shall be used to denote the genitive case.

5207 **ab_alt_mon** Define the abbreviated month names, corresponding to the `%Ob` conversion
5208 specification. The operand shall consist of twelve <semicolon>-separated
5209 strings, each surrounded by double-quotes. The first string shall be the
5210 abbreviated name of the first month of the year (January), the second the
5211 abbreviated name of the second month, and so on. For languages having both
5212 a genitive (when used with a day number) and a nominative (no day number)
5213 case, this operand shall be used to denote the nominative case.

5214 **mon** Define the full month names, corresponding to the `%B` conversion
5215 specification. The operand shall consist of twelve <semicolon>-separated
5216 strings, each surrounded by double-quotes. The first string shall be the full
5217 name of the first month of the year (January), the second the full name of the
5218 second month, and so on. For languages having both a genitive (when used
5219 with a day number) and a nominative (no day number) case, this operand
5220 shall be used to denote the genitive case.

5221 **alt_mon** Define the full month names, corresponding to the `%OB` conversion
5222 specification. The operand shall consist of twelve <semicolon>-separated
5223 strings, each surrounded by double-quotes. The first string shall be the full
5224 name of the first month of the year (January), the second the full name of the
5225 second month, and so on. For languages having both a genitive (when used
5226 with a day number) and a nominative (no day number) case, this operand
5227 shall be used to denote the nominative case.

5228 **d_t_fmt** Define the appropriate date and time representation, corresponding to the `%c`
5229 conversion specification. The operand shall consist of a string containing any
5230 combination of characters and conversion specifications. In addition, the
5231 string can contain escape sequences defined in the table in Table 5-1 (on page
5232 113) (`'\\'`,`'\a'`,`'\b'`,`'\f'`,`'\n'`,`'\r'`,`'\t'`,`'\v'`).

d_fmt Define the appropriate date representation, corresponding to the `%x` conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in Table 5-1 (on page 113).

t_fmt Define the appropriate time representation, corresponding to the `%X` conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in Table 5-1 (on page 113).

am_pm Define the appropriate representation of the *ante-meridiem* and *post-meridiem* strings, corresponding to the `%p` conversion specification. The operand shall consist of two strings, separated by a <semicolon>, each surrounded by double-quotes; the first string shall represent the *ante-meridiem* designation, the last string the *post-meridiem* designation. If and only if the 12-hour format is not supported in the locale, both strings shall be empty.

t_fmt_ampm Define the appropriate time representation in the 12-hour clock format with **am_pm**, corresponding to the `%r` conversion specification. The operand shall consist of a string and can contain any combination of characters and conversion specifications. If and only if the 12-hour format is not supported in the locale, the string shall be empty.

era Define how years are counted and displayed for each era in a locale. The operand shall consist of <semicolon>-separated strings. Each string shall be an era description segment with the format:

direction`:`*offset*`:`*start_date*`:`*end_date*`:`*era_name*`:`*era_format*

according to the definitions below. There can be as many era description segments as are necessary to describe the different eras.

Note: The start of an era might not be the earliest point in the era—it may be the latest. For example, the Christian era BC starts on the day before January 1, AD 1, and increases with earlier time.

direction Either a `'+'` or a `'-'` character. The `'+'` character shall indicate that years closer to the *start_date* have lower numbers than those closer to the *end_date*. The `'-'` character shall indicate that years closer to the *start_date* have higher numbers than those closer to the *end_date*.

offset The number of the year closest to the *start_date* in the era, corresponding to the `%Ey` conversion specification.

start_date A date in the form *yyyy*/*mm*/*dd*, where *yyyy*, *mm*, and *dd* are the year, month, and day numbers respectively of the start of the era. Years prior to AD 1 shall be represented as negative numbers.

end_date The ending date of the era, in the same format as the *start_date*, or one of the two special values `"-*"` or `"+*"`. The value `"-*"` shall indicate that the ending date is the beginning of time. The value `"+*"` shall indicate that the ending date is the end of time.

era_name A string representing the name of the era, corresponding to the `%EC` conversion specification.

era_format A string for formatting the year in the era, corresponding to the `%EY` conversion specification.

era_d_fmt Define the format of the date in alternative era notation, corresponding to the `%Ex` conversion specification.

era_t_fmt Define the locale's appropriate alternative time format, corresponding to the `%EX` conversion specification.

era_d_t_fmt Define the locale's appropriate alternative date and time format, corresponding to the `%Ec` conversion specification.

alt_digits Define alternative symbols for digits, corresponding to the `%O` modified conversion specification. The operand shall consist of <semicolon>-separated strings, each surrounded by double-quotes. The first string shall be the alternative symbol corresponding with zero, the second string the symbol corresponding with one, and so on. Up to 100 alternative symbol strings can be specified. The `%O` modifier shall indicate that the string corresponding to the value specified via the conversion specification shall be used instead of the value.

7.3.5.2 *LC_TIME C-Language Access*

The following constants used to identify items of *langinfo* data can be used as arguments to the *nl_langinfo*() function to access information in the *LC_TIME* category. These constants are defined in the **<langinfo.h>** header.

ABDAY_*x* The abbreviated weekday names (for example, Sun), where *x* is a number from 1 to 7.

DAY_*x* The full weekday names (for example, Sunday), where *x* is a number from 1 to 7.

ABMON_*x* The abbreviated month names (for example, Jan), where *x* is a number from 1 to 12.

ABALTMON_*x* The alternative abbreviated month names (for example, Jan), where *x* is a number from 1 to 12.

MON_*x* The full month names (for example, January), where *x* is a number from 1 to 12.

ALTMON_*x* The alternative full month names (for example, January), where *x* is a number from 1 to 12.

D_T_FMT The appropriate date and time representation.

D_FMT The appropriate date representation.

T_FMT The appropriate time representation.

AM_STR The appropriate ante-meridiem affix; if AM_STR and PM_STR are both empty strings, the 12-hour format is not supported in the locale.

PM_STR The appropriate post-meridiem affix; if AM_STR and PM_STR are both empty strings, the 12-hour format is not supported in the locale.

T_FMT_AMPM The appropriate time representation in the 12-hour clock format; if the 12-hour format is not supported in the locale, this shall be either an empty string or a string specifying a 24-hour clock format.

ERA The era description segments, which describe how years are counted and displayed for each era in a locale. Each era description segment shall have the format:

5322 *direction*:*offset*:*start_date*:*end_date*:*era_name*:*era_format*

5323 according to the definitions below. There can be as many era description
5324 segments as are necessary to describe the different eras. Era description
5325 segments are separated by <semicolon> characters.

5326 *direction* Either a '+' or a '−' character. The '+' character shall indicate
5327 that years closer to the *start_date* have lower numbers than those
5328 closer to the *end_date*. The '−' character shall indicate that years
5329 closer to the *start_date* have higher numbers than those closer to
5330 the *end_date*.

5331 *offset* The number of the year closest to the *start_date* in the era.

5332 *start_date* A date in the form *yyyy*/*mm*/*dd*, where *yyyy*, *mm*, and *dd* are the
5333 year, month, and day numbers respectively of the start of the era.
5334 Years prior to AD 1 shall be represented as negative numbers.

5335 *end_date* The ending date of the era, in the same format as the *start_date*,
5336 or one of the two special values "−*" or "+*". The value "−*"
5337 shall indicate that the ending date is the beginning of time. The
5338 value "+*" shall indicate that the ending date is the end of time.

5339 *era_name* The era, corresponding to the %EC conversion specification.

5340 *era_format* The format of the year in the era, corresponding to the %EY
5341 conversion specification.

5342 ERA_D_FMT The era date format.

5343 ERA_T_FMT The locale's appropriate alternative time format, corresponding to the %EX
5344 conversion specification.

5345 ERA_D_T_FMT The locale's appropriate alternative date and time format, corresponding to
5346 the %Ec conversion specification.

5347 ALT_DIGITS The alternative symbols for digits, corresponding to the %O conversion
5348 specification modifier. The value consists of <semicolon>-separated symbols.
5349 The first is the alternative symbol corresponding to zero, the second is the
5350 symbol corresponding to one, and so on. Up to 100 alternative symbols may
5351 be specified.

5352 *7.3.5.3 LC_TIME Category in the POSIX Locale*

5353 The *LC_TIME* category definition of the POSIX locale follows; the code listing depicts the
5354 *localedef* input; the table represents the same information with the addition of *localedef* keywords,
5355 conversion specifiers used by the *date* utility and the *strftime*(), *wcsftime*(), and *strptime*()
5356 functions, and *nl_langinfo*() constants.

```
5357 LC_TIME
5358 # This is the POSIX locale definition for
5359 # the LC_TIME category.
5360 #
5361 # Abbreviated weekday names (%a)
5362 abday      "<S><u><n>";"<M><o><n>";"<T><u><e>";"<W><e><d>";\
5363            "<T><h><u>";"<F><r><i>";"<S><a><t>"
5364 #
5365 # Full weekday names (%A)
5366 day        "<S><u><n><d><a><y>";"<M><o><n><d><a><y>";\
```



```
                "<T><u><e><s><d><a><y>";"<W><e><d><n><e><s><d><a><y>";\
                "<T><h><u><r><s><d><a><y>";"<F><r><i><d><a><y>";\
                "<S><a><t><u><r><d><a><y>"
        #
        # Abbreviated month names (%b)
        abmon       "<J><a><n>";"<F><e><b>";"<M><a><r>";\
                    "<A><p><r>";"<M><a><y>";"<J><u><n>";\
                    "<J><u><l>";"<A><u><g>";"<S><e><p>";\
                    "<O><c><t>";"<N><o><v>";"<D><e><c>"
        #
        # Full month names (%B)
        mon         "<J><a><n><u><a><r><y>";"<F><e><b><r><u><a><r><y>";\
                    "<M><a><r><c><h>";"<A><p><r><i><l>";\
                    "<M><a><y>";"<J><u><n><e>";\
                    "<J><u><l><y>";"<A><u><g><u><s><t>";\
                    "<S><e><p><t><e><m><b><e><r>";"<O><c><t><o><b><e><r>";\
                    "<N><o><v><e><m><b><e><r>";"<D><e><c><e><m><b><e><r>"
        #
        # Equivalent of AM/PM (%p)      "AM";"PM"
        am_pm       "<A><M>";"<P><M>"
        #
        # Appropriate date and time representation (%c)
        #    "%a %b %e %H:%M:%S %Y"
        d_t_fmt     "<percent-sign><a><space><percent-sign><b>\
        <space><percent-sign><e><space><percent-sign><H>\
        <colon><percent-sign><M><colon><percent-sign><S>\
        <space><percent-sign><Y>"
        #
        # Appropriate date representation (%x)   "%m/%d/%y"
        d_fmt       "<percent-sign><m><slash><percent-sign><d>\
        <slash><percent-sign><y>"
        #
        # Appropriate time representation (%X)   "%H:%M:%S"
        t_fmt       "<percent-sign><H><colon><percent-sign><M>\
        <colon><percent-sign><S>"
        #
        # Appropriate 12-hour time representation (%r) "%I:%M:%S %p"
        t_fmt_ampm "<percent-sign><I><colon><percent-sign><M><colon>\
        <percent-sign><S><space><percent-sign><p>"
        #
        END LC_TIME
```

localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
d_t_fmt	D_T_FMT	%c	"%a %b %e %H:%M:%S %Y"
d_fmt	D_FMT	%x	"%m/%d/%y"
t_fmt	T_FMT	%X	"%H:%M:%S"
am_pm	AM_STR	%p	"AM"
am_pm	PM_STR	%p	"PM"
t_fmt_ampm	T_FMT_AMPM	%r	"%I:%M:%S %p"
day	DAY_1	%A	"Sunday"
day	DAY_2	%A	"Monday"

localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
day	DAY_3	%A	"Tuesday"
day	DAY_4	%A	"Wednesday"
day	DAY_5	%A	"Thursday"
day	DAY_6	%A	"Friday"
day	DAY_7	%A	"Saturday"
abday	ABDAY_1	%a	"Sun"
abday	ABDAY_2	%a	"Mon"
abday	ABDAY_3	%a	"Tue"
abday	ABDAY_4	%a	"Wed"
abday	ABDAY_5	%a	"Thu"
abday	ABDAY_6	%a	"Fri"
abday	ABDAY_7	%a	"Sat"
mon	MON_1	%B	"January"
mon	MON_2	%B	"February"
mon	MON_3	%B	"March"
mon	MON_4	%B	"April"
mon	MON_5	%B	"May"
mon	MON_6	%B	"June"
mon	MON_7	%B	"July"
mon	MON_8	%B	"August"
mon	MON_9	%B	"September"
mon	MON_10	%B	"October"
mon	MON_11	%B	"November"
mon	MON_12	%B	"December"
alt_mon	ALTMON_1	%OB	N/A
alt_mon	ALTMON_2	%OB	N/A
alt_mon	ALTMON_3	%OB	N/A
alt_mon	ALTMON_4	%OB	N/A
alt_mon	ALTMON_5	%OB	N/A
alt_mon	ALTMON_6	%OB	N/A
alt_mon	ALTMON_7	%OB	N/A
alt_mon	ALTMON_8	%OB	N/A
alt_mon	ALTMON_9	%OB	N/A
alt_mon	ALTMON_10	%OB	N/A
alt_mon	ALTMON_11	%OB	N/A
alt_mon	ALTMON_12	%OB	N/A
abmon	ABMON_1	%b	"Jan"
abmon	ABMON_2	%b	"Feb"
abmon	ABMON_3	%b	"Mar"
abmon	ABMON_4	%b	"Apr"
abmon	ABMON_5	%b	"May"
abmon	ABMON_6	%b	"Jun"
abmon	ABMON_7	%b	"Jul"
abmon	ABMON_8	%b	"Aug"
abmon	ABMON_9	%b	"Sep"
abmon	ABMON_10	%b	"Oct"
abmon	ABMON_11	%b	"Nov"
abmon	ABMON_12	%b	"Dec"
ab_alt_mon	ABALTMON_1	%Ob	N/A
ab_alt_mon	ABALTMON_2	%Ob	N/A

localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
ab_alt_mon	ABALTMON_3	%Ob	N/A
ab_alt_mon	ABALTMON_4	%Ob	N/A
ab_alt_mon	ABALTMON_5	%Ob	N/A
ab_alt_mon	ABALTMON_6	%Ob	N/A
ab_alt_mon	ABALTMON_7	%Ob	N/A
ab_alt_mon	ABALTMON_8	%Ob	N/A
ab_alt_mon	ABALTMON_9	%Ob	N/A
ab_alt_mon	ABALTMON_10	%Ob	N/A
ab_alt_mon	ABALTMON_11	%Ob	N/A
ab_alt_mon	ABALTMON_12	%Ob	N/A
era	ERA	%EC, %Ey, %EY	N/A
era_d_fmt	ERA_D_FMT	%Ex	N/A
era_t_fmt	ERA_T_FMT	%EX	N/A
era_d_t_fmt	ERA_D_T_FMT	%Ec	N/A
alt_digits	ALT_DIGITS	%O	N/A

The entry N/A indicates the value is not available in the POSIX locale.

7.3.6 LC_MESSAGES

The *LC_MESSAGES* category shall define the format and values used by various utilities for affirmative and negative responses. This information is available through the *nl_langinfo*( ) function.

The message catalog used by the standard utilities and selected by the *catopen*( ) function shall be determined by the setting of *NLSPATH*; see Chapter 8 (on page 167). The *LC_MESSAGES* category can be specified as part of an *NLSPATH* substitution field.

The following keywords shall be recognized as part of the locale definition file.

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

Note: This is a *localedef* keyword, unavailable through *nl_langinfo*( ).

yesexpr The operand consists of an extended regular expression (see Section 9.4, on page 187) that describes acceptable affirmative responses to a question expecting an affirmative or negative response.

noexpr The operand consists of an extended regular expression that describes acceptable negative responses to a question expecting an affirmative or negative response.

7.3.6.1 *LC_MESSAGES Category in the POSIX Locale*

The format and values for affirmative and negative responses of the POSIX locale follow; the code listing depicting the *localedef* input, the table representing the same information with the addition of *nl_langinfo*( ) constants.

```
LC_MESSAGES
# This is the POSIX locale definition for
# the LC_MESSAGES category.
#
yesexpr "<circumflex><left-square-bracket><y><Y><right-square-bracket>"
#
```



```
noexpr  "<circumflex><left-square-bracket><n><N><right-square-bracket>"
#
END LC_MESSAGES
```

localedef Keyword	langinfo Constant	POSIX Locale Value
yesexpr	YESEXPR	`"^[yY]"`
noexpr	NOEXPR	`"^[nN]"`

7.4 Locale Definition Grammar

The grammar and lexical conventions in this section shall together describe the syntax for the locale definition source. The general conventions for this style of grammar are described in XCU Section 1.3 (on page 2461). The grammar shall take precedence over the text in this chapter.

7.4.1 Locale Lexical Conventions

The lexical conventions for the locale definition grammar are described in this section.

The following tokens shall be processed (in addition to those string constants shown in the grammar):

LOC_NAME A string of characters representing the name of a locale.

CHAR Any single character.

NUMBER A decimal number, represented by one or more decimal digits.

COLLSYMBOL A symbolic name, enclosed between angle brackets. The string cannot duplicate any charmap symbol defined in the current charmap (if any), or a **COLLELEMENT** symbol.

COLLELEMENT A symbolic name, enclosed between angle brackets, which cannot duplicate either any charmap symbol or a **COLLSYMBOL** symbol.

CHARCLASS A string of alphanumeric characters from the portable character set, the first of which is not a digit, consisting of at least one and at most {CHARCLASS_NAME_MAX} bytes, and optionally surrounded by double-quotes.

CHARSYMBOL A symbolic name, enclosed between angle brackets, from the current charmap (if any).

OCTAL_CHAR One or more octal representations of the encoding of each byte in a single character. The octal representation consists of an escape character (normally a <backslash>) followed by two or more octal digits.

HEX_CHAR One or more hexadecimal representations of the encoding of each byte in a single character. The hexadecimal representation consists of an escape character followed by the constant *x* and two or more hexadecimal digits.

DECIMAL_CHAR One or more decimal representations of the encoding of each byte in a single character. The decimal representation consists of an escape character followed by a character `'d'` and two or more decimal digits.

ELLIPSIS The string "...".

EXTENDED_REG_EXP An extended regular expression as defined in the grammar in Section 9.5 (on page 191).

EOL The line termination character <newline>.

7.4.2 Locale Grammar

This section presents the grammar for the locale definition.

```
%token              LOC_NAME
%token              CHAR
%token              NUMBER
%token              COLLSYMBOL COLLELEMENT
%token              CHARSYMBOL OCTAL_CHAR HEX_CHAR DECIMAL_CHAR
%token              ELLIPSIS
%token              EXTENDED_REG_EXP
%token              EOL

%start              locale_definition

%%

locale_definition   : global_statements locale_categories
                    |                   locale_categories
                    ;

global_statements   : global_statements symbol_redefine
                    | symbol_redefine
                    ;

symbol_redefine     : 'escape_char'  CHAR EOL
                    | 'comment_char' CHAR EOL
                    ;

locale_categories   : locale_categories locale_category
                    | locale_category
                    ;

locale_category     : lc_ctype | lc_collate | lc_messages
                    | lc_monetary | lc_numeric | lc_time
                    ;

/* The following grammar rules are common to all categories */

char_list           : char_list char_symbol
                    | char_symbol
                    ;

char_symbol         : CHAR | CHARSYMBOL
                    | OCTAL_CHAR | HEX_CHAR | DECIMAL_CHAR
                    ;

elem_list           : elem_list char_symbol
                    | elem_list COLLSYMBOL
                    | elem_list COLLELEMENT
                    | char_symbol
                    | COLLSYMBOL
```



```
                        | COLLELEMENT
                        ;

symb_list               : symb_list COLLSYMBOL
                        | COLLSYMBOL
                        ;

locale_name             : LOC_NAME
                        | '"' LOC_NAME '"'
                        ;

/* The following is the LC_CTYPE category grammar */

lc_ctype                : ctype_hdr ctype_keywords         ctype_tlr
                        | ctype_hdr 'copy' locale_name EOL ctype_tlr
                        ;

ctype_hdr               : 'LC_CTYPE' EOL
                        ;

ctype_keywords          : ctype_keywords ctype_keyword
                        | ctype_keyword
                        ;

ctype_keyword           : charclass_keyword charclass_list EOL
                        | charconv_keyword charconv_list EOL
                        | 'charclass' charclass_namelist EOL
                        ;

charclass_namelist      : charclass_namelist ';' CHARCLASS
                        | CHARCLASS
                        ;

charclass_keyword       : 'upper' | 'lower' | 'alpha' | 'digit'
                        | 'punct' | 'xdigit' | 'space' | 'print'
                        | 'graph' | 'blank' | 'cntrl' | 'alnum'
                        | CHARCLASS
                        ;

charclass_list          : charclass_list ';' char_symbol
                        | charclass_list ';' ELLIPSIS ';' char_symbol
                        | char_symbol
                        ;

charconv_keyword        : 'toupper'
                        | 'tolower'
                        ;

charconv_list           : charconv_list ';' charconv_entry
                        | charconv_entry
                        ;

charconv_entry          : '(' char_symbol ',' char_symbol ')'
                        ;

ctype_tlr               : 'END' 'LC_CTYPE' EOL
                        ;

/* The following is the LC_COLLATE category grammar */

lc_collate              : collate_hdr collate_keywords       collate_tlr
```



```
                        | collate_hdr 'copy' locale_name EOL collate_tlr
                        ;

collate_hdr             : 'LC_COLLATE' EOL
                        ;

collate_keywords        :                order_statements
                        | opt_statements order_statements
                        ;

opt_statements          : opt_statements collating_symbols
                        | opt_statements collating_elements
                        | collating_symbols
                        | collating_elements
                        ;

collating_symbols       : 'collating-symbol' COLLSYMBOL EOL
                        ;

collating_elements      : 'collating-element' COLLELEMENT
                        | 'from' '"' elem_list '"' EOL
                        ;

order_statements        : order_start collation_order order_end
                        ;

order_start             : 'order_start' EOL
                        | 'order_start' order_opts EOL
                        ;

order_opts              : order_opts ';' order_opt
                        | order_opt
                        ;

order_opt               : order_opt ',' opt_word
                        | opt_word
                        ;

opt_word                : 'forward' | 'backward' | 'position'
                        ;

collation_order         : collation_order collation_entry
                        | collation_entry
                        ;

collation_entry         : COLLSYMBOL EOL
                        | collation_element weight_list EOL
                        | collation_element             EOL
                        ;

collation_element       : char_symbol
                        | COLLELEMENT
                        | ELLIPSIS
                        | 'UNDEFINED'
                        ;

weight_list             : weight_list ';' weight_symbol
                        | weight_list ';'
                        | weight_symbol
                        ;
```



```
5688        weight_symbol       : /* empty */
5689                            | char_symbol
5690                            | COLLSYMBOL
5691                            | '"' elem_list '"'
5692                            | '"' symb_list '"'
5693                            | ELLIPSIS
5694                            | 'IGNORE'
5695                            ;
5696        order_end           : 'order_end' EOL
5697                            ;
5698        collate_tlr         : 'END' 'LC_COLLATE' EOL
5699                            ;
5700        /* The following is the LC_MESSAGES category grammar */
5701        lc_messages         : messages_hdr messages_keywords      messages_tlr
5702                            | messages_hdr 'copy' locale_name EOL messages_tlr
5703                            ;
5704        messages_hdr        : 'LC_MESSAGES' EOL
5705                            ;
5706        messages_keywords   : messages_keywords messages_keyword
5707                            | messages_keyword
5708                            ;
5709        messages_keyword    : 'yesexpr' '"' EXTENDED_REG_EXP '"' EOL
5710                            | 'noexpr'  '"' EXTENDED_REG_EXP '"' EOL
5711                            ;
5712        messages_tlr        : 'END' 'LC_MESSAGES' EOL
5713                            ;
5714        /* The following is the LC_MONETARY category grammar */
5715        lc_monetary         : monetary_hdr monetary_keywords       monetary_tlr
5716                            | monetary_hdr 'copy' locale_name EOL  monetary_tlr
5717                            ;
5718        monetary_hdr        : 'LC_MONETARY' EOL
5719                            ;
5720        monetary_keywords   : monetary_keywords monetary_keyword
5721                            | monetary_keyword
5722                            ;
5723        monetary_keyword    : mon_keyword_string mon_string EOL
5724                            | mon_keyword_char NUMBER EOL
5725                            | mon_keyword_char '-1'   EOL
5726                            | mon_keyword_grouping mon_group_list EOL
5727                            ;
5728        mon_keyword_string  : 'int_curr_symbol' | 'currency_symbol'
5729                            | 'mon_decimal_point' | 'mon_thousands_sep'
5730                            | 'positive_sign' | 'negative_sign'
5731                            ;
5732        mon_string          : '"' char_list '"'
```



```
                        | '"'
                        ;
mon_keyword_char        : 'int_frac_digits' | 'frac_digits'
                        | 'p_cs_precedes' | 'p_sep_by_space'
                        | 'n_cs_precedes' | 'n_sep_by_space'
                        | 'p_sign_posn' | 'n_sign_posn'
                        | 'int_p_cs_precedes' | 'int_p_sep_by_space'
                        | 'int_n_cs_precedes' | 'int_n_sep_by_space'
                        | 'int_p_sign_posn' | 'int_n_sign_posn'
                        ;
mon_keyword_grouping : 'mon_grouping'
                        ;
mon_group_list          : NUMBER
                        | mon_group_list ';' NUMBER
                        ;
monetary_tlr            : 'END' 'LC_MONETARY' EOL
                        ;
/* The following is the LC_NUMERIC category grammar */
lc_numeric              : numeric_hdr numeric_keywords       numeric_tlr
                        | numeric_hdr 'copy' locale_name EOL numeric_tlr
                        ;
numeric_hdr             : 'LC_NUMERIC' EOL
                        ;
numeric_keywords        : numeric_keywords numeric_keyword
                        | numeric_keyword
                        ;
numeric_keyword         : num_keyword_string num_string EOL
                        | num_keyword_grouping num_group_list EOL
                        ;
num_keyword_string      : 'decimal_point'
                        | 'thousands_sep'
                        ;
num_string              : '"' char_list '"'
                        | '"'
                        ;
num_keyword_grouping: 'grouping'
                        ;
num_group_list          : NUMBER
                        | num_group_list ';' NUMBER
                        ;
numeric_tlr             : 'END' 'LC_NUMERIC' EOL
                        ;
/* The following is the LC_TIME category grammar */
lc_time                 : time_hdr time_keywords          time_tlr
                        | time_hdr 'copy' locale_name EOL time_tlr
```



```
                        ;

time_hdr                : 'LC_TIME' EOL
                        ;

time_keywords           : time_keywords time_keyword
                        | time_keyword
                        ;

time_keyword            : time_keyword_name time_list EOL
                        | time_keyword_fmt time_string EOL
                        | time_keyword_opt time_list EOL
                        ;

time_keyword_name       : 'abday' | 'day' | 'abmon' | 'mon'
                        ;

time_keyword_fmt        : 'd_t_fmt' | 'd_fmt' | 't_fmt'
                        | 'am_pm' | 't_fmt_ampm'
                        ;

time_keyword_opt        : 'era' | 'era_d_fmt' | 'era_t_fmt'
                        | 'era_d_t_fmt' | 'alt_digits'
                        | 'ab_alt_mon' | 'alt_mon'
                        ;

time_list               : time_list ';' time_string
                        | time_string
                        ;

time_string             : '"' char_list '"'
                        ;

time_tlr                : 'END' 'LC_TIME' EOL
                        ;
```


5804 Chapter 8

5805 # Environment Variables

5806 ## 8.1 Environment Variable Definition

5807 Environment variables defined in this chapter affect the operation of multiple utilities, functions,
5808 and applications. There are other environment variables that are of interest only to specific
5809 utilities. Environment variables that apply to a single utility only are defined as part of the
5810 utility description. See the ENVIRONMENT VARIABLES section of the utility descriptions in
5811 the Shell and Utilities volume of POSIX.1-2024 for information on environment variable usage.

5812 The value of an environment variable is an arbitrary sequence of bytes, except for the null byte.
5813 For a C-language program, an array of strings called the environment shall be made available
5814 when a process begins. The array is pointed to by the external variable *environ*, which is defined
5815 as:

5816 ```
extern char **environ;
```

5817 These strings have the form *name*=*value*; *name*s shall not contain any bytes that have the encoded
5818 value of the character '='. For values to be portable across systems conforming to
5819 POSIX.1-2024, the value shall be composed of bytes that have the encoded value of characters
5820 from the portable character set (except NUL and as indicated below). There is no meaning
5821 associated with the order of strings in the environment. If more than one string in an
5822 environment of a process has the same *name*, the consequences are undefined.

5823 Environment variable names used by the utilities in the Shell and Utilities volume of
5824 POSIX.1-2024 consist solely of uppercase letters, digits, and the <underscore> ('_') from the
5825 characters defined in Table 6-1 (on page 117) and do not begin with a digit. Other characters, and
5826 byte sequences that do not form valid characters, may be permitted by an implementation;
5827 applications shall tolerate the presence of such names. Uppercase and lowercase letters shall
5828 retain their unique identities and shall not be folded together. The name space of environment
5829 variable names containing lowercase letters is reserved for applications. Applications can define
5830 any environment variables with names from this name space without modifying the behavior of
5831 the standard utilities.

5832 **Note:** Other applications may have difficulty dealing with environment variable names that start with
5833 a digit. For this reason, use of such names is not recommended anywhere.

5834 The *values* that the environment variables may be assigned are not restricted except that they are
5835 considered to end with a null byte and the total space used to store the environment and the
5836 arguments to the process is limited to {ARG_MAX} bytes.

5837 Other *name*=*value* pairs may be placed in the environment by, for example, calling any of the
5838 XSI *setenv*(), *unsetenv*(), or *putenv*() functions, assigning a new value to the *environ* variable, or by
5839 using *envp* arguments when creating a process; see *exec* in the System Interfaces volume of
5840 POSIX.1-2024.

5841 If the application modifies the pointers to which *environ* points, the behavior of all interfaces
5842 described in the System Interfaces volume of POSIX.1-2024 is undefined.

5843 It is unwise to conflict with certain variables that are frequently exported by widely used
5844 command interpreters and applications:

  
```

ARFLAGS	*IFS*	*MAILPATH*	*PS1*
CC	*LANG*	*MAILRC*	*PS2*
CDPATH	*LC_ALL*	*MAKEFLAGS*	*PS3*
CFLAGS	*LC_COLLATE*	*MAKESHELL*	*PS4*
CHARSET	*LC_CTYPE*	*MANPATH*	*PWD*
COLUMNS	*LC_MESSAGES*	*MBOX*	*RANDOM*
DATEMSK	*LC_MONETARY*	*MORE*	*SECONDS*
DEAD	*LC_NUMERIC*	*MSGVERB*	*SHELL*
EDITOR	*LC_TIME*	*NLSPATH*	*TERM*
ENV	*LDFLAGS*	*NPROC*	*TERMCAP*
EXINIT	*LEX*	*OLDPWD*	*TERMINFO*
FC	*LFLAGS*	*OPTARG*	*TMPDIR*
FCEDIT	*LINENO*	*OPTERR*	*TZ*
FFLAGS	*LINES*	*OPTIND*	*USER*
GET	*LISTER*	*PAGER*	*VISUAL*
GFLAGS	*LOGNAME*	*PATH*	*YACC*
HISTFILE	*LPDEST*	*PPID*	*YFLAGS*
HISTORY	*MAIL*	*PRINTER*	
HISTSIZE	*MAILCHECK*	*PROCLANG*	
HOME	*MAILER*	*PROJECTDIR*	

Additionally, a subset of the above variables are manipulated by shell built-in utilities outside of shell assignments. If an attempt is made to mark any of the following variables as *readonly*, then either the *readonly* utility shall reject the attempt, or *readonly* shall succeed but the shell can still modify the variables outside of assignment context, or *readonly* shall succeed but use of a shell built-in that would otherwise modify such a variable shall fail.

LINENO
OLDPWD
OPTARG
OPTIND
PWD

Implementations may provide an implementation-defined set of additional variables which are manipulated by implementation-specific built-in utilities not defined in this standard. The *readonly* utility shall not reject marking these additional variables as *readonly*, but when marked *readonly*, those extension utilities shall either continue to modify the variables, or shall fail because the variable is *readonly*. None of the variables defined by this standard shall be in this implementation-defined set.

If the variables in the following two sections are present in the environment during the execution of an application or utility, they shall be given the meaning described below. Some are placed into the environment by the implementation at the time the user logs in; all can be added or changed by the user or any ancestor of the current process. The implementation adds or changes environment variables named in POSIX.1-2024 only as specified in POSIX.1-2024. If they are defined in the application's environment, the utilities in the Shell and Utilities volume of POSIX.1-2024 and the functions in the System Interfaces volume of POSIX.1-2024 assume they have the specified meaning. Conforming applications shall not set these environment variables to have meanings other than as described. See *getenv*( ) (on page 1120) and XCU Section 2.13 (on page 2522) for methods of accessing these variables.

Implementations may ignore some environment variables at the point of use for security reasons, for example in programs whose real and effective user IDs or real and effective group IDs were not equal at program startup. The behavior shall be as if the implementation obtains the values for these environment variables using *secure_getenv*( ) instead of *getenv*( ) (see

getenv( )); they shall not be removed from the environment of affected processes and shall be inherited as required by this standard.

8.2 Internationalization Variables

This section describes environment variables that are relevant to the operation of internationalized interfaces described in POSIX.1-2024.

Users may use the following environment variables to announce specific localization requirements to applications. Applications can retrieve this information using the *setlocale*( ) function to initialize the correct behavior of the internationalized interfaces. The descriptions of the internationalization environment variables describe the resulting behavior only when the application locale is initialized in this way. The use of the internationalization variables by utilities described in the Shell and Utilities volume of POSIX.1-2024 is described in the ENVIRONMENT VARIABLES section for those utilities in addition to the global effects described in this section.

LANG This variable shall determine the locale category for native language, local customs, and coded character set in the absence of the *LC_ALL* and other *LC_** (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*) environment variables. This can be used by applications to determine the language to use for error messages and instructions, collating sequences, date formats, and so on.

LANGUAGE

The *LANGUAGE* environment variable shall be examined to determine the messages object to be used for the *gettext* family of functions or the *gettext* and *ngettext* utilities XSI if *NLSPATH* is not set or the evaluation of *NLSPATH* did not lead to a suitable messages object being found. The value of *LANGUAGE* shall be a list of locale names separated by a <colon> (`':'`) character. If *LANGUAGE* is set to a non-empty string, each locale name shall be tried in the specified order and if a messages object is found, it shall be used for translation. If a locale name has the format *language*[*_territory*][*.codeset*][*@modifier*], additional searches of locale names without *.codeset* (if present), without *_territory* (if present), and without *@modifier* (if present) may be performed; if *.codeset* is not present, additional searches of locale names with an added *.codeset* may be performed. If locale names contain a <slash> (`'/'`) character, or consist entirely of a dot (`"."`) or dot-dot (`".."`) character sequence, or are empty the behavior is implementation defined and they may be ignored for security reasons.

The locale names in *LANGUAGE* shall override the locale name associated with the ``active category'' of the current locale or, in the case of functions with an *_l* suffix, the provided locale object, and the language-specific part of the default search path for messages objects, unless the locale name that would be overridden is C or POSIX. For the *dcgettext*( ), *dcgettext_l*( ), *dcngettext*( ), and *dcngettext_l*( ) functions, the active category is specified by the *category* argument; for all other *gettext* family functions and for the *gettext* and *ngettext* utilities, the active category is *LC_MESSAGES*.

For example, if:

- The *LC_MESSAGES* environment variable is `"de_DE"` (and *LC_ALL* is unset) and `setlocale(LC_ALL, "")` has been used to set the current locale

- The *LANGUAGE* environment variable is `"fr_FR:it"`
- Messages objects are by default searched for in **/gettextlib**

then the following pathnames are tried in this order by *gettext* family functions that have neither a category argument nor an *_l* suffix until a valid messages object is found:

- **/gettextlib/fr_FR/LC_MESSAGES/***textdomain***.mo**
- (Optionally) **/gettextlib/fr/LC_MESSAGES/***textdomain***.mo**
- (Optionally) the above two pathnames with added *.codeset* elements
- **/gettextlib/it/LC_MESSAGES/***textdomain***.mo**
- (Optionally) the above pathname with added *.codeset* elements
- **/gettextlib/de_DE/LC_MESSAGES/***textdomain***.mo**

LC_ALL This variable shall determine the values for all locale categories. The value of the *LC_ALL* environment variable has precedence over any of the other environment variables starting with *LC_* (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*) and the *LANG* environment variable.

LC_COLLATE
This variable shall determine the locale category for character collation. It determines collation information for regular expressions and sorting, including equivalence classes and multi-character collating elements, in various utilities and the *strcoll*() and *strxfrm*() functions. Additional semantics of this variable, if any, are implementation-defined.

LC_CTYPE This variable shall determine the locale category for character handling functions, such as *tolower*(), *toupper*(), and *isalpha*(). This environment variable determines the interpretation of sequences of bytes of text data as characters (for example, single as opposed to multi-byte characters), the classification of characters (for example, alpha, digit, graph), and the behavior of character classes. Additional semantics of this variable, if any, are implementation-defined.

LC_MESSAGES
This variable shall determine the locale category for processing affirmative and negative responses and the language and cultural conventions in which messages should be written. It also affects the behavior of the *catopen*() function in determining the message catalog. Additional semantics of this variable, if any, are implementation-defined. The language and cultural conventions of diagnostic and informative messages whose format is unspecified by POSIX.1-2024 should be affected by the setting of *LC_MESSAGES*.

LC_MONETARY
This variable shall determine the locale category for monetary-related numeric formatting information. Additional semantics of this variable, if any, are implementation-defined.

LC_NUMERIC
This variable shall determine the locale category for numeric formatting (for example, thousands separator and radix character) information in various utilities as well as the formatted I/O operations in *printf*() and *scanf*() and the string conversion functions in *strtod*(). Additional semantics of this variable, if any, are implementation-defined.

LC_TIME This variable shall determine the locale category for date and time formatting information. It affects the behavior of the time functions in *strftime*( ). Additional semantics of this variable, if any, are implementation-defined.

XSI *NLSPATH* This variable shall contain a sequence of templates to be used by *catopen*( ) when attempting to locate message catalogs, and by the *gettext* family of functions when locating messages objects. Each template consists of an optional prefix, one or more conversion specifications, and an optional suffix.

The conversion specification descriptions below refer to a ``currently active text domain''. The currently active text domain is, in decreasing order of precedence:

- The *domain* parameter of the *gettext* family of functions or the *gettext* and *ngettext* utilities
- The text domain bound by the last call to *textdomain*( ) when using a *gettext* family function, or the *TEXTDOMAIN* environment variable when using the *gettext* and *ngettext* utilities
- The default text domain

Conversion specifications consist of a `'%'` symbol, followed by a single-letter keyword. The following conversion specifications are currently defined:

`%N` The value of the *name* parameter passed to *catopen*( ) or the currently active text domain of the *gettext* family of functions and the *gettext* and *ngettext* utilities (see above).

`%L` The locale name given by the value of the active category (see *LANGUAGE* above) in either the current locale or, in the case of functions with an *_l* suffix, the provided locale object.

`%l` The *language* element of the locale name that would result from a `%L` conversion.

`%t` The *territory* element of the locale name that would result from a `%L` conversion.

`%c` The *codeset* element of the locale name that would result from a `%L` conversion.

`%%` A single `'%'` character.

An empty string shall be substituted if the specified value is not currently defined. The separators <underscore> (`'_'`) and <period> (`'.'`) shall not be included in the `%t` and `%c` conversion specifications.

Templates defined in *NLSPATH* are separated by <colon> characters (`':'`). A leading, trailing, or two adjacent <colon> characters (`"::"`) shall be equivalent to specifying `%N`.

Since <colon> is a separator in this context, directory names that might be used in *NLSPATH* should not include a <colon> character.

Example 1, for an application that uses *catopen*( ) but does not use the *gettext* family of functions:

```
NLSPATH="/system/nlslib/%N.cat"
```

indicates that *catopen*( ) should look for all message catalogs in the directory **/system/nlslib**, where the catalog name should be constructed from the *name* argument (replacing `%N`) passed to *catopen*( ), with the suffix **.cat**.

Example 2, for an application that uses the *gettext* family of functions but does not use *catopen*( ):

```
NLSPATH="/usr/lib/locale/fr/LC_MESSAGES/%N.mo"
```

indicates that the *gettext* family of functions (and the *gettext* and *ngettext* utilities) should look for all messages objects in the directory **/usr/lib/locale/fr/LC_MESSAGES**, where the messages object's name should be constructed from the currently active text domain (replacing `%N`), with the suffix **.mo**.

Example 3, for an application that uses *catopen*( ) but does not use the *gettext* family of functions:

```
NLSPATH=":%N.cat:/nlslib/%L/%N.cat"
```

indicates that *catopen*( ) should look for the requested message catalog in *name*, *name***.cat**, and **/nlslib/***localename***/***name***.cat**, where *localename* is the locale name given by the value of the *LC_MESSAGES* category of the current locale.

Example 4, for an application that uses the *gettext* family of functions but does not use *catopen*( ):

```
NLSPATH="/usr/lib/locale/%L/%N.mo:/usr/lib/locale/fr/%N.mo"
```

indicates that the *gettext* family of functions (and the *gettext* and *ngettext* utilities) should look for all messages objects first in **/usr/lib/locale/***localename***/***textdomain***.mo**, and if not found there, then try in **/usr/lib/locale/fr/***textdomain***.mo**, where *localename* is the locale name given by the value of the active category in the current locale or provided locale object.

Example 5, for an application that uses *catopen*( ) and the *gettext* family of functions:

```
NLSPATH="/usr/lib/locale/%L/%N.mo:/system/nlslib/%L/%N.cat"
```

indicates that the *gettext* family of functions (and the *gettext* and *ngettext* utilities) should look for all messages objects in **/usr/lib/locale/***localename***/***textdomain***.mo**, where *localename* is the locale name given by the value of the active category in the current locale or provided locale object. Also, *catopen*( ) should look for all message catalogs in the directory **/system/nlslib/***localename***/***name***.cat**, (assuming that **/usr/lib/locale/***localename***/***name***.mo** is not a message catalog). In this scenario, *catopen*( ) ignores all files that are not valid message catalogs while traversing *NLSPATH*. Furthermore, the *gettext* family of functions and the *gettext* and *ngettext* utilities ignore all files that are not valid messages objects found while traversing *NLSPATH*.

Users should not set the *NLSPATH* variable unless they have a specific reason to override the default system path. Setting *NLSPATH* to override the default system path may produce undefined results in the standard utilities other than *gettext* and *ngettext*, and in applications with appropriate privileges.

Specifying a relative pathname in the *NLSPATH* environment variable should be avoided without a specific reason, including the use of a leading, trailing, or two adjacent <colon> characters, since it may result in messages objects being searched for in a directory relative to the current working directory of the calling process; if the process calls the *chdir*( ) function, the directory searched for may also be changed.

TEXTDOMAIN
: Specify the text domain name that the *gettext* and *ngettext* utilities use during the search for messages objects. This is identical to the messages object filename without the **.mo** suffix.

TEXTDOMAINDIR
: Specify the pathname to the root directory of the messages object hierarchy the *gettext* and *ngettext* utilities use during the search for messages objects. If present, it shall replace the default root directory pathname. *NLSPATH* has precedence over *TEXTDOMAINDIR*.

The environment variables *LANG*, *LC_ALL*, *LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*, and *NLSPATH* provide for the support of internationalized applications. The standard utilities shall make use of these environment variables as described in this section and the individual ENVIRONMENT VARIABLES sections for the utilities. See Section 7.1 (on page 127) for the consequences of setting these variables to locales with different character sets.

The values of locale categories shall be determined by a precedence order; the first condition met below determines the value:

1. If the *LC_ALL* environment variable is defined and is not null, the value of *LC_ALL* shall be used.
2. If the *LC_** environment variable (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*) is defined and is not null, the value of the environment variable shall be used to initialize the category that corresponds to the environment variable.
3. If the *LANG* environment variable is defined and is not null, the value of the *LANG* environment variable shall be used.
4. Otherwise, the implementation-defined default locale shall be used.

If the locale value is `"C"` or `"POSIX"`, the POSIX locale shall be used and the standard utilities behave in accordance with the rules in Section 7.2 (on page 128) for the associated category.

If the locale value begins with a <slash>, it shall be interpreted as the pathname of a file that was created in the output format used by the *localedef* utility; see OUTPUT FILES under *localedef*. Referencing such a pathname shall result in that locale being used for the indicated category.

XSI If the locale value has the form:

language`[`*_territory*`][`*.codeset*`]`

it refers to an implementation-provided locale, where settings of language, territory, and codeset are implementation-defined.

LC_COLLATE, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, and *LC_TIME* are defined to accept an additional field *@modifier*, which allows the user to select a specific instance of localization data within a single category (for example, for selecting the dictionary as opposed to the character ordering of data). The syntax for these environment variables is thus defined as:

`[`*language*`[`*_territory*`][`*.codeset*`][@`*modifier*`]]`

For example, if a user wanted to interact with the system in French, but required to sort German text files, *LANG* and *LC_COLLATE* could be defined as:

```
LANG=Fr_FR
LC_COLLATE=De_DE
```


This could be extended to select dictionary collation (say) by use of the *@modifier* field; for example:

```
LC_COLLATE=De_DE@dict
```

An implementation may support other formats.

If the locale value is not recognized by the implementation, the behavior is unspecified.

These environment variables are used by the *newlocale*( ) and *setlocale*( ) functions, and by the standard utilities.

Additional criteria for determining a valid locale name are implementation-defined.

8.3 Other Environment Variables

COLUMNS This variable shall represent a decimal integer >0 used to indicate the user's preferred width in column positions for the terminal screen or window; see Section 3.75 (on page 42). If this variable is unset or null, the number of columns shall be set according to the terminal window size (see XSH *tcgetwinsize*( )); if the terminal window size cannot be obtained, the implementation determines the number of columns, appropriate for the terminal or window, in an unspecified manner. When *COLUMNS* is set, the number of columns in the terminal window size and any terminal-width information implied by *TERM* are overridden. Users and conforming applications should not set *COLUMNS* unless they wish to override the system selection and produce output unrelated to the terminal characteristics.

Users should not need to set this variable in the environment unless there is a specific reason to override the implementation's default behavior, such as to display data in an area arbitrarily smaller than the terminal or window.

XSI *DATEMSK* Indicates the pathname of the template file used by *getdate*( ).

HOME The system shall initialize this variable at the time of login to be a pathname of the user's home directory. See **<pwd.h>**.

LINES This variable shall represent a decimal integer >0 used to indicate the user's preferred number of lines on a page or the vertical screen or window size in lines. A line in this case is a vertical measure large enough to hold the tallest character in the character set being displayed. If this variable is unset or null, the number of lines shall be set either to the number of rows in the terminal window size (see XSH *tcgetwinsize*( )) or to a smaller number if appropriate for the terminal or window (for example, if the terminal baud rate is low); if the terminal window size cannot be obtained, the implementation determines the number of lines, appropriate for the terminal or window, in an unspecified manner. When *LINES* is set, the number of rows in the terminal window size and any terminal-height information implied by *TERM* are overridden. Users and conforming applications should not set *LINES* unless they wish to override the system selection and produce output unrelated to the terminal characteristics.

Users should not need to set this variable in the environment unless there is a specific reason to override the implementation's default behavior, such as to display data in an area arbitrarily smaller than the terminal or window.

LOGNAME The system shall initialize this variable at the time of login to be the user's login name. See **<pwd.h>**. For a value of *LOGNAME* to be portable across implementations of POSIX.1-2024, the value should be composed of characters from the portable filename character set.

XSI *MSGVERB* Describes which message components shall be used in writing messages by *fmtmsg*( ).

PATH This variable shall represent the sequence of path prefixes that certain functions and utilities apply in searching for an executable file. The prefixes shall be separated by a <colon> (`':'`). If the pathname being sought contains no <slash> (`'/'`) characters, and hence is a filename, the list shall be searched from beginning to end, applying the filename to each prefix and attempting to resolve the resulting pathname (see Section 4.16, on page 105), until an executable file with appropriate execution permissions is found. When a non-zero-length prefix is applied to this filename, a <slash> shall be inserted between the prefix and the filename if the prefix did not end in <slash>. A zero-length prefix is a legacy feature that indicates the current working directory. It appears as two adjacent <colon> characters (`"::"`), as an initial <colon> preceding the rest of the list, or as a trailing <colon> following the rest of the list. A strictly conforming application shall use an actual pathname (such as **.**) to represent the current working directory in *PATH*. If the pathname being sought contains any <slash> characters, the search through the path prefixes shall not be performed and the pathname shall be resolved as described in Section 4.16 (on page 105). If *PATH* is unset or is set to null, or if a path prefix in *PATH* contains a <percent-sign> character (`'%'`), the path search is implementation-defined.

Since <colon> is a separator in this context, directory names that might be used in *PATH* should not include a <colon> character. Since <percent-sign> may have an implementation-defined meaning when searching for built-in utilities, directory names in *PATH* to be used to search for non-built-in utilities should not contain a <percent-sign> character.

PWD This variable shall represent an absolute pathname of the current working directory. It shall not contain any components that are dot or dot-dot. The value is set by the *cd* utility, and by the *sh* utility during initialization.

SHELL This variable shall represent a pathname of the user's preferred command language interpreter. If this interpreter does not conform to the Shell Command Language in XCU Chapter 2 (on page 2472), utilities may behave differently from those described in POSIX.1-2024.

TMPDIR This variable shall represent a pathname of a directory made available for programs that need a place to create temporary files.

TERM This variable shall represent the terminal type for which output is to be prepared. This information is used by utilities and application programs wishing to exploit special capabilities specific to a terminal. The format and allowable values of this environment variable are unspecified.

TZ This variable shall represent timezone information. The contents of the environment variable named *TZ* shall be used by the *ctime*( ), *localtime*( ), *localtime_r*( ), *strftime*( ), and *mktime*( ) functions, and by various utilities, to override the default timezone. The application shall ensure that the value of *TZ* is in one of the three formats (spaces inserted for clarity):

6208 `:`*characters*

6209 or:

6210 *std offset dst offset*`,` *rule*

6211 or:

6212 A format specifying a geographical timezone or a special timezone.

6213 If *TZ* is of the first format (that is, if the first character is a <colon>), the
6214 characters following the <colon> are handled in an implementation-defined
6215 manner.

6216 The expanded form of the second format (without the inserted spaces) is as
6217 follows:

6218 *stdoffset*`[`*dst*`[`*offset*`][,`*start*`[/`*time*`],`*end*`[/`*time*`]]]`

6219 Where:

6220 *std* and *dst* Indicate no less than three, nor more than {TZNAME_MAX},
6221 bytes that are the designation for the standard (*std*) or the
6222 Daylight Saving (*dst*) timezone. Only *std* is required; if *dst* is
6223 missing, then Daylight Saving Time does not apply in this locale.

6224 **Note:** The usage of the terms ``Standard Time'' and ``Daylight
6225 Saving Time'' is not necessarily related to any legislated
6226 timezone.

6227 Each of these fields may occur in either of two formats quoted or
6228 unquoted:

6229 — In the quoted form, the first character shall be the <less-
6230 than-sign> (`'<'`) character and the last character shall be
6231 the <greater-than-sign> (`'>'`) character. All characters
6232 between these quoting characters shall be alphanumeric
6233 characters from the portable character set in the current
6234 locale, the <plus-sign> (`'+'`) character, or the <hyphen-
6235 minus> (`'−'`) character. The *std* and *dst* fields in this case
6236 shall not include the quoting characters and the quoting
6237 characters do not contribute to the three byte minimum
6238 length and {TZNAME_MAX} maximum length.

6239 — In the unquoted form, all characters in these fields shall be
6240 alphabetic characters from the portable character set in the
6241 current locale.

6242 The interpretation of *std* and, if present, *dst* is unspecified if the
6243 field is less than three bytes or more than {TZNAME_MAX}
6244 bytes, or if it contains characters other than those specified.

6245 *offset* Indicates the value added to the local time to arrive at
6246 Coordinated Universal Time. The *offset* has the form:

6247 *hh*`[:`*mm*`[:`*ss*`]]`

6248 The minutes (*mm*) and seconds (*ss*) are optional. The hour (*hh*)
6249 shall be required and may be a single digit. The *offset* following
6250 *std* shall be required. If no *offset* follows *dst*, Daylight Saving
6251 Time is assumed to be one hour ahead of standard time. One or
6252 more digits may be used; the value is always interpreted as a

6253 decimal number. The hour shall be between zero and 24, and the
6254 minutes (and seconds)—if present—between zero and 59. The
6255 result of using values outside of this range is unspecified. If
6256 preceded by a `'-'`, the timezone shall be east of the Prime
6257 Meridian; otherwise, it shall be west (which may be indicated by
6258 an optional preceding `'+'`).

6259 *rule* Indicates when to change from standard time to Daylight Saving
6260 Time, and when to change back. The *rule* has the form:

6261 *date*`[/`*time*`],`*date*`[/`*time*`]`

6262 where the first *date* describes when the change from standard
6263 time to Daylight Saving Time occurs and the second *date*
6264 describes when it ends; if the second *date* is specified as earlier in
6265 the year than the first, then the year begins and ends in Daylight
6266 Saving Time. Each *time* field describes when, in current local
6267 time, the change to the other time is made.

6268 The format of *date* is one of the following:

6269 J*n* The Julian day *n* ($1 \le n \le 365$). Leap days shall not be
6270 counted. That is, in all years—including leap years—
6271 February 28 is day 59 and March 1 is day 60. It is
6272 impossible to refer explicitly to the occasional February
6273 29.

6274 *n* The zero-based Julian day ($0 \le n \le 365$). Leap days shall
6275 be counted, and it is possible to refer to February 29.

6276 M*m.n.d* The *d*'th day ($0 \le d \le 6$) of week *n* of month *m* of the
6277 year ($1 \le n \le 5$, $1 \le m \le 12$, where week 5 means ``the last
6278 *d* day in month *m*'' which may occur in either the fourth
6279 or the fifth week). Week 1 is the first week in which the
6280 *d*'th day occurs. Day zero is Sunday.

6281 The *time* has the same format as *offset* except that the hour can
6282 range from zero to 167. If preceded by a `'-'`, the time shall
6283 count backwards before midnight. For example, `"47:30"`
6284 stands for 23:30 the next day, and `"-3:30"` stands for 20:30 the
6285 previous day. The default, if *time* is not given, shall be 02:00:00.

6286 Daylight Saving Time is in effect all year if it starts January 1 at 00:00 and ends
6287 December 31 at 24:00 plus the difference between Daylight Saving Time and
6288 standard time, leaving no room for standard time in the calendar. For
6289 example, `TZ='EST5EDT,0/0,J365/25'` represents a time zone that
6290 observes Daylight Saving Time all year, being 4 hours west of UTC with
6291 abbreviation `"EDT"`.

6292 If the *dst* field is specified and the *rule* field is not, it is implementation-defined
6293 when the changes to and from Daylight Saving Time occur.

6294 If *TZ* is of the third format (that is, if the first character is not a <colon> and
6295 the value does not match the syntax for the second format), the value indicates
6296 either a geographical timezone or a special timezone from an implementation-
6297 defined timezone database. Typically these take the form

6298 *Area/Location*

as in the IANA timezone database. Examples of geographical timezones that may be supported include `Africa/Cairo`, `America/New_York`, `America/Indiana/Indianapolis`, `Asia/Tokyo`, and `Europe/London`. The data for each geographical timezone shall include:

- The offset from Coordinated Universal Time of the timezone's standard time.
- If Daylight Saving Time (DST) is, or has historically been, observed: a method to discover the dates and times of transitions to and from DST and the offset from Coordinated Universal Time during periods when DST was, is, or is predicted to be, in effect.
- The timezone names for standard time (*std*) and, if observed, for DST (*dst*) to be used by *tzset*( ). These shall each contain no more than {TZNAME_MAX} bytes.

If there are any historical variations, or known future variations, of the above data for a geographical timezone, these variations shall be included in the database, except that historical variations from before the Epoch need not be included.

If the database incorporates an external database such as the one maintained by IANA, the implementation shall provide an implementation-defined method to allow the database to be updated, for example the method specified by RFC 6557.

Chapter 9

Regular Expressions

Regular Expressions (REs) provide a mechanism to select specific strings from a set of character strings.

Regular expressions are a context-independent syntax that can represent a wide variety of character sets and character set orderings, where these character sets are interpreted according to the current locale. While many regular expressions can be interpreted differently depending on the current locale, many features, such as character class expressions, provide for contextual invariance across locales.

The Basic Regular Expression (BRE) notation and construction rules in Section 9.3 (on page 181) shall apply to most utilities supporting regular expressions. Some utilities, instead, support the Extended Regular Expressions (ERE) described in Section 9.4 (on page 187); any exceptions for both cases are noted in the descriptions of the specific utilities using regular expressions. Both BREs and EREs are supported by the Regular Expression Matching interface in the System Interfaces volume of POSIX.1-2024 under *regcomp*( ), *regexec*( ), and related functions.

9.1 Regular Expression Definitions

For the purposes of this section, the following definitions shall apply:

entire regular expression
The concatenated set of one or more BREs or EREs that make up the pattern specified for string selection.

escape sequence
The escape character followed by any single character, which is thereby ``escaped''. The escape character is a <backslash> that is neither in a bracket expression nor itself escaped.

leftmost
The characters closest to the beginning of the string.

matched
A sequence of zero or more characters shall be said to be matched by a BRE or ERE when the characters in the sequence correspond to a sequence of characters defined by the pattern.

Matching shall be based on the bit pattern used for encoding the character, not on the graphic representation of the character. This means that if a character set contains two or more encodings for a graphic symbol, or if the strings searched contain text encoded in more than one codeset, no attempt is made to search for any other representation of the encoded symbol. If that is required, the user can specify equivalence classes containing all variations of the desired graphic symbol.

The search for a matching sequence starts at the beginning of a string and stops when the first sequence matching the expression is found, where ``first'' is defined to mean ``begins earliest in the string''. If the pattern permits a variable number of matching characters and thus there is more than one such sequence starting at that point, the longest such sequence is matched. For example, the BRE `"bb*"` matches the second to fourth characters of the

string "abbbc", and the ERE "(wee|week)(knights|night)" matches all ten characters of the string "weeknights".

Consistent with the whole match being the longest of the leftmost matches, each subpattern, from left to right, shall match the longest possible string. For this purpose, a null string shall be considered to be longer than no match at all. For example, matching the BRE "\(.*\).*" against "abcdef", the subexpression "(\1)" is "abcdef", and matching the BRE "\(a*\)*" against "bc", the subexpression "(\1)" is the null string. However, matching the ERE "(.*?).*" against "abcdef", the subpattern "(.*?)" matches the empty string, since that is the longest possible match for the ERE ".*?".

When a multi-character collating element in a bracket expression (see Section 9.3.5, on page 182) is involved, the longest sequence shall be measured in characters consumed from the string to be matched; that is, the collating element counts not as one element, but as the number of characters it matches.

BRE (ERE) matching a single character
A BRE or ERE that shall match either a single character or a single collating element.

Only a BRE or ERE of this type that includes a bracket expression (see Section 9.3.5, on page 182) can match a collating element.

BRE (ERE) matching multiple characters
A BRE or ERE that shall match a concatenation of single characters or collating elements.

Such a BRE or ERE is made up from a BRE (ERE) matching a single character and BRE (ERE) special characters.

invalid
This section uses the term ``invalid'' for certain constructs or conditions. Invalid REs shall cause the utility or function using the RE to generate an error condition. When invalid is not used, violations of the specified syntax or semantics for REs produce undefined results: this may entail an error, enabling an extended syntax for that RE, or using the construct in error as literal characters to be matched. For example, the BRE construct "\{1,2,3\}" does not comply with the grammar. A conforming application cannot rely on it producing an error nor matching the literal characters "\{1,2,3\}".

9.2 Regular Expression General Requirements

The requirements in this section shall apply to both basic and extended regular expressions.

The use of regular expressions is generally associated with text processing. REs (BREs and EREs) operate on text strings; that is, zero or more characters followed by an end-of-string delimiter (typically NUL). Some utilities employing regular expressions limit the processing to lines; that is, zero or more characters followed by a <newline>.

In the functions processing regular expressions described in System Interfaces volume of POSIX.1-2024, the <newline> is regarded as an ordinary character and both a <period> and a non-matching list can match one. The Shell and Utilities volume of POSIX.1-2024 specifies within the individual descriptions of those standard utilities employing regular expressions whether they permit matching of <newline> characters; if not stated otherwise, the use of literal <newline> characters or any escape sequence equivalent in either patterns or matched text produces undefined results. Those utilities (like *grep*) that do not allow <newline> characters to match are responsible for eliminating any <newline> from strings before matching against the RE. The *regcomp*() function in the System Interfaces volume of POSIX.1-2024, however, can provide support for such processing without violating the rules of this section.

The interfaces specified in POSIX.1-2024 do not permit the inclusion of a NUL character in an RE or in the string to be matched. If during the operation of a standard utility a NUL is included in the text designated to be matched, that NUL may designate the end of the text string for the purposes of matching.

Some standard utilities and functions support case-insensitive regular expression matching. When this type of matching is in effect, the matching process shall be modified as described in Section 4.1 (on page 95).

The implementation shall support any regular expression that does not exceed 256 bytes in length.

9.3 Basic Regular Expressions

9.3.1 BREs Matching a Single Character or Collating Element

When not inside a bracket expression, the following shall match a single character:

- a BRE ordinary character
- a BRE special character or `']'` preceded by an unescaped <backslash>
- a <period>

A bracket expression shall match a single character or a single collating element.

9.3.2 BRE Ordinary Characters

An ordinary character is a BRE that matches itself: any character in the supported character set, except for the BRE special characters listed in Section 9.3.3 (on page 182).

When not inside a bracket expression, the interpretation of an ordinary character preceded by an unescaped <backslash> is undefined, except for:

- The characters `')'`, `'('`, `'{'`, and `'}'`
- The digits 1 to 9 inclusive (see Section 9.3.6, on page 185)
- The `']'` character; `"\]"` shall match a `']'` character
- The `'?'`, `'+'`, and `'|'` characters; it is implementation-defined whether `"\?"`, `"\+"`, and `"\|"` each match the literal character `'?'`, `'+'`, or `'|'`, respectively, or behave as described for the ERE special characters `'?'`, `'+'`, and `'|'`, respectively (see Section 9.4.3, on page 188).

  Note: A future version of this standard may require `"\?"`, `"\+"`, and `"\|"` to behave as described for the ERE special characters `'?'`, `'+'`, and `'|'`, respectively.

9.3.3 BRE Special Characters

A BRE special character has special properties in certain contexts. Outside those contexts, or when preceded by an unescaped <backslash>, such a character is a BRE that matches the special character itself. The BRE special characters and the contexts in which they have their special meaning are as follows:

`.[\` The <period>, <left-square-bracket>, and <backslash> shall be special except when used in a bracket expression (see Section 9.3.5). An expression containing a `'['` that is unescaped and is not part of a bracket expression produces undefined results.

`*` The <asterisk> shall be special except when used:

- In a bracket expression
- As the first character of an entire BRE (after an initial `'^'`, if any)
- Immediately following a `"\|"` escape sequence (after an initial `'^'`, if any), if the implementation does not match the escape sequence `"\|"` to the literal character `'|'`.
- As the first character of a subexpression (after an initial `'^'`, if any); see Section 9.3.6 (on page 185)

`^` The <circumflex> shall be special when used as an anchor (see Section 9.3.8, on page 186). The <circumflex> shall signify a non-matching list expression when it occurs first in a list, immediately following a <left-square-bracket> (see Section 9.3.5).

`$` The <dollar-sign> shall be special when used as an anchor.

9.3.4 Periods in BREs

When not inside a bracket expression, a <period> (`'.'`) is a BRE that shall match any character in the supported character set except NUL.

9.3.5 RE Bracket Expression

A bracket expression (an expression enclosed in square brackets, `"[]"`) is an RE that shall match a specific set of single characters, and may match a specific set of multi-character collating elements, based on the non-empty set of list expressions contained in the bracket expression.

The following rules and definitions apply to bracket expressions:

1. A bracket expression is either a matching list expression or a non-matching list expression. It consists of one or more expressions: ordinary characters, collating elements, collating symbols, equivalence classes, character classes, or range expressions. The <right-square-bracket> (`']'`) shall lose its special meaning and represent itself in a bracket expression if it occurs first in the list (after an initial <circumflex> (`'^'`), if any). Otherwise, it shall terminate the bracket expression, unless it appears in a collating symbol (such as `"[.].]"`) or is the ending <right-square-bracket> for a collating symbol, equivalence class, or character class. When the bracket expression appears within a BRE, the special characters `'.'`, `'*'`, `'['`, and `'\\'` (<period>, <asterisk>, <left-square-bracket>, and <backslash>, respectively) shall lose their special meaning within the bracket expression. When the bracket expression appears within an ERE, the special characters `'.'`, `'('`, `'*'`, `'+'`, `'?'`, `'{'`, `'|'`, `'$'`, `'['`, and `'\\'` (<period>, <left-parenthesis>, <asterisk>, <plus-sign>, <question-mark>, <left-brace>, <vertical-line>, <dollar-sign>, <left-square-bracket>, and <backslash>, respectively) shall lose their

special meaning within the bracket expression; <circumflex> (`'^'`) shall lose its special meaning as an anchor. When the bracket expression appears within a shell pattern (see XCU Section 2.14, on page 2523), the special characters `'?'`, `'*'`, and `'['` (<question-mark>, <asterisk>, and <left-square-bracket>, respectively) shall lose their special meaning within the bracket expression; whether or not <backslash> (`'\\'`) loses its special meaning as a pattern matching character is described in XCU Section 2.14.1 (on page 2523), but in contexts where a shell-quoting <backslash> can be used it shall retain its special meaning (see XCU Section 2.2, on page 2472). For example:

```
$ ls
! $ - \ a b c
$ echo [a\-c]
- a c
$ echo [\!a]
! a
$ echo ["!\$a-c"]
! $ - a c
$ echo [!"\$a-c"]
! \ b
$ echo [!\]\\]
! $ - a b c
```

The character sequences `"[."`, `"[="`, and `"[:"` (<left-square-bracket> followed by a <period>, <equals-sign>, or <colon>) shall be special inside a bracket expression and are used to delimit collating symbols, equivalence class expressions, and character class expressions. These symbols shall be followed by a valid expression and the matching terminating sequence `".]"`, `"=]"`, or `":]"`, as described in the following items.

2. A matching list expression specifies a list that shall match any single character that is matched by one of the expressions represented in the list. The first character in the list cannot be the <circumflex>. An ordinary character in the list shall only match that character; for example, `"[abc]"` is an RE that only matches one of the characters `'a'`, `'b'`, or `'c'`.

   It is unspecified whether a matching list expression matches a multi-character collating element that is matched by one of the expressions.

3. A non-matching list expression begins with a <circumflex> (`'^'`), and the matching behavior shall be the logical inverse of the corresponding matching list expression (the same bracket expression but without the leading <circumflex>). For example, since the RE `"[abc]"` only matches `'a'`, `'b'`, or `'c'`, it follows that `"[^abc]"` is an RE that matches any character except `'a'`, `'b'`, or `'c'`. It is unspecified whether a non-matching list expression matches a multi-character collating element that is not matched by any of the expressions. The <circumflex> shall have this special meaning only when it occurs first in the list, immediately following the <left-square-bracket>.

4. A collating symbol is a collating element enclosed within bracket-period (`"[."` and `".]"`) delimiters. Collating elements are defined as described in Section 7.3.2.4 (on page 142). Conforming applications shall represent multi-character collating elements as collating symbols when it is necessary to distinguish them from a list of the individual characters that make up the multi-character collating element. For example, if the string `"ch"` is a collating element defined using the line:

   ```
   collating-element <ch-digraph> from "<c><h>"
   ```

   in the locale definition, the expression `"[[.ch.]]"` shall be treated as an RE containing the collating symbol `'ch'`, while `"[ch]"` shall be treated as an RE matching `'c'` or `'h'`.

Collating symbols are recognized only inside bracket expressions. If the string is not a collating element in the current locale, the expression is invalid.

5. An equivalence class expression shall represent the set of collating elements belonging to an equivalence class, as described in Section 7.3.2.4 (on page 142). Only primary equivalence classes shall be recognized. The class shall be expressed by enclosing any one of the collating elements in the equivalence class within bracket-equal (`"[="` and `"=]"`) delimiters. For example, if `'a'`, `'à'`, and `'^'` belong to the same equivalence class, then `"[[=a=]b]"`, `"[[=à=]b]"`, and `"[[=^=]b]"` are each equivalent to `"[aà^b]"`. If the collating element does not belong to an equivalence class, the equivalence class expression shall be treated as a collating symbol.

6. A character class expression shall represent the union of two sets:

   a. The set of single characters that belong to the character class, as defined in the *LC_CTYPE* category in the current locale.

   b. An unspecified set of multi-character collating elements.

   All character classes specified in the current locale shall be recognized. A character class expression is expressed as a character class name enclosed within bracket-<colon> (`"[:"` and `":]"`) delimiters.

   The following character class expressions shall be supported in all locales:

```
[:alnum:]   [:cntrl:]   [:lower:]   [:space:]
[:alpha:]   [:digit:]   [:print:]   [:upper:]
[:blank:]   [:graph:]   [:punct:]   [:xdigit:]
```

   In addition, character class expressions of the form:

   `[:`*name*`:]`

   are recognized in those locales where the *name* keyword has been given a **charclass** definition in the *LC_CTYPE* category.

7. In the POSIX locale, a range expression represents the set of collating elements that fall between two elements in the collation sequence, inclusive. In other locales, a range expression has unspecified behavior: strictly conforming applications shall not rely on whether the range expression is valid, or on the set of collating elements matched. A range expression shall be expressed as the starting point and the ending point separated by a <hyphen-minus> (`'-'`).

   In the following, all examples assume the POSIX locale.

   The starting range point and the ending range point shall be a collating element or collating symbol. An equivalence class expression used as a starting or ending point of a range expression produces unspecified results. An equivalence class can be used portably within a bracket expression, but only outside the range. If the represented set of collating elements is empty, it is unspecified whether the expression matches nothing, or is treated as invalid.

   The interpretation of range expressions where the ending range point is also the starting range point of a subsequent range expression (for example, `"[a-m-o]"`) is undefined.

   The <hyphen-minus> character shall be treated as itself if it occurs first (after an initial `'^'`, if any) or last in the list, or as an ending range point in a range expression. As examples, the expressions `"[-ac]"` and `"[ac-]"` are equivalent and match any of the characters `'a'`, `'c'`, or `'-'`; `"[^-ac]"` and `"[^ac-]"` are equivalent and match any characters except `'a'`, `'c'`, or `'-'`; the expression `"[%--]"` matches any of the

characters between '%' and '−' inclusive; the expression "[−−@]" matches any of the characters between '−' and '@' inclusive; and the expression "[a−−@]" is either invalid or equivalent to '@', because the letter 'a' follows the symbol '−' in the POSIX locale. To use a <hyphen-minus> as the starting range point, it shall either come first in the bracket expression or be specified as a collating symbol; for example, "[][.-.]−0]", which matches either a <right-square-bracket> or any character or collating element that collates between <hyphen-minus> and 0, inclusive.

If a bracket expression specifies both '−' and ']', the ']' shall be placed first (after the '^', if any) and the '−' last within the bracket expression.

8. If a bracket expression contains at least three list elements, where the first and last list elements are the same single-character element of <period>, <equals-sign>, or <colon>, then it is unspecified whether the bracket expression will be treated as a collating symbol, equivalence class, or character class, respectively; treated as a matching list expression; or treated as an invalid bracket expression.

9.3.6 BREs Matching Multiple Characters

The following rules can be used to construct BREs matching multiple characters from BREs matching a single character:

1. The concatenation of BREs shall match the concatenation of the strings matched by each component of the BRE.

2. A subexpression can be defined within a BRE by enclosing it between the character pairs "\(" and "\)". Such a subexpression shall match whatever it would have matched without the "\(" and "\)", except that anchoring within subexpressions is optional behavior; see Section 9.3.8 (on page 186). Subexpressions can be arbitrarily nested.

3. The back-reference expression '\n' shall match the same (possibly empty) string of characters as was matched by a subexpression enclosed between "\(" and "\)" preceding the '\n'. The character 'n' shall be a digit from 1 through 9, specifying the *n*th subexpression (the one that begins with the *n*th "\(" from the beginning of the pattern and ends with the corresponding paired "\)"). The expression is invalid if less than *n* subexpressions precede the '\n'. The string matched by a contained subexpression shall be within the string matched by the containing subexpression. If the containing subexpression does not match, or if there is no match for the contained subexpression within the string matched by the containing subexpression, then back-reference expressions corresponding to the contained subexpression shall not match. When a subexpression matches more than one string, a back-reference expression corresponding to the subexpression shall refer to the last matched string. For example, the expression "^\(.*\)\1$" matches strings consisting of two adjacent appearances of the same substring, and the expression "\(a\)*\1" fails to match 'a', the expression "\(a\(b\)*\)*\2" fails to match 'abab', and the expression "^\(ab*\)*\1$" matches 'ababbabb', but fails to match 'ababbab'.

4. When a BRE matching a single character, a subexpression, or a back-reference is followed by the special character <asterisk> ('*'), together with that <asterisk> it shall match what zero or more consecutive occurrences of the BRE would match. For example, "[ab]*" and "[ab][ab]" are equivalent when matching the string "ab".

5. When a BRE matching a single character, a subexpression, or a back-reference is followed by an interval expression of the format "\{m\}", "\{m,\}", or "\{m,n\}", together with that interval expression it shall match what repeated consecutive occurrences of the BRE would match. The values of *m* and *n* are decimal integers in the range 0

≤*m*≤*n*≤{RE_DUP_MAX}, where *m* specifies the exact or minimum number of occurrences and *n* specifies the maximum number of occurrences. The expression `"\{m\}"` shall match exactly *m* occurrences of the preceding BRE, `"\{m,\}"` shall match at least *m* occurrences, and `"\{m,n\}"` shall match any number of occurrences between *m* and *n*, inclusive.

For example, in the string `"abababccccccd"` the BRE `"c\{3\}"` is matched by characters seven to nine, the BRE `"\(ab\)\{4,\}"` is not matched at all, and the BRE `"c\{1,3\}d"` is matched by characters ten to thirteen.

The behavior of multiple adjacent duplication symbols (`'*'` and intervals) produces undefined results.

A subexpression repeated by an <asterisk> (`'*'`) or an interval expression shall not match a null expression unless this is the only match for the repetition or it is necessary to satisfy the exact or minimum number of occurrences for the interval expression.

9.3.7 BRE Precedence

The order of precedence shall be as shown in the following table:

BRE Precedence (from high to low)	
Collation-related bracket symbols	`[==] [::] [..]`
Escaped characters	`\<special character>`
Bracket expression	`[]`
Subexpressions/back-references	`\(\) \n`
Single-character-BRE duplication	`* \{m,n\}`
Concatenation	
Anchoring	`^ $`

9.3.8 BRE Expression Anchoring

A BRE can be limited to matching expressions that begin or end a string; this is called ``anchoring''. The <circumflex> and <dollar-sign> special characters shall be considered BRE anchors in the following contexts:

1. A <circumflex> (`'^'`) shall be an anchor when used as the first character of an entire BRE and, if the implementation does not match the escape sequence `"\|"` to the literal character `'|'`, when used immediately following a `"\|"` escape sequence that is not inside a subexpression. The implementation may also treat a <circumflex> as an anchor when used inside a subexpression; in this case it shall be an anchor only when either of the following is true:

   - It is the first character of the subexpression.
   - It immediately follows a `"\|"` escape sequence and the implementation does not match the escape sequence `"\|"` to the literal character `'|'`.

   The <circumflex> shall anchor the expression (or optionally subexpression) to the beginning of a string; only sequences starting at the first character of a string shall be matched by the BRE. For example, the BRE `"^ab"` matches `"ab"` in the string `"abcdef"`, but fails to match in the string `"cdefab"`. The BRE `"\(^ab\)"` may match the former string. A portable BRE shall escape a leading <circumflex> in a subexpression to match a literal <circumflex>.

2. A <dollar-sign> (`'$'`) shall be an anchor when used as the last character of an entire BRE and, if the implementation does not match the escape sequence `"\|"` to the literal character `'|'`, when used immediately preceding a `"\|"` escape sequence that is not inside a subexpression. The implementation may also treat a <dollar-sign> as an anchor when used inside a subexpression; in this case it shall be an anchor only when either of the following is true:

   - It is the last character of the subexpression.
   - It immediately precedes a `"\|"` escape sequence and the implementation does not match the escape sequence `"\|"` to the literal character `'|'`.

   The <dollar-sign> shall anchor the expression (or optionally subexpression) to the end of the string being matched; the <dollar-sign> can be said to match the end-of-string following the last character. A portable BRE shall escape a trailing <dollar-sign> in a subexpression to match a literal <dollar-sign>.

3. A BRE anchored by both `'^'` and `'$'` shall match only an entire string. For example, the BRE `"^abcdef$"` matches strings consisting only of `"abcdef"`.

9.4 Extended Regular Expressions

The extended regular expression (ERE) notation and construction rules shall apply to utilities defined as using extended regular expressions; any exceptions to the following rules are noted in the descriptions of the specific utilities using EREs.

9.4.1 EREs Matching a Single Character or Collating Element

When not inside a bracket expression, the following shall match a single character:

- an ERE ordinary character
- an ERE special character, `']'`, or `'}'` preceded by an unescaped <backslash>
- a <period>

A bracket expression shall match a single character or a single collating element. An ERE matching a single character enclosed in parentheses shall match the same as the ERE without parentheses would have matched.

9.4.2 ERE Ordinary Characters

An ordinary character is an ERE that matches itself. An ordinary character is any character in the supported character set, except for the ERE special characters listed in Section 9.4.3 (on page 188). When not inside a bracket expression, the interpretation of an ordinary character preceded by an unescaped <backslash> is undefined, except for the `']'` and `'}'` characters; `"\]"` and `"\}"` shall match the `']'` and `'}'` characters, respectively.

9.4.3 ERE Special Characters

An ERE special character has special properties in certain contexts. Outside those contexts, or when preceded by an unescaped <backslash>, such a character shall be an ERE that matches the special character itself. The extended regular expression special characters and the contexts in which they shall have their special meaning are as follows:

`. [ \ (` The <period>, <left-square-bracket>, <backslash>, and <left-parenthesis> shall be special except when used in a bracket expression (see Section 9.3.5, on page 182). When not inside a bracket expression, an unescaped <left-parenthesis> immediately followed by a <right-parenthesis> produces undefined results. A <left-square-bracket> that is unescaped and is not part of a bracket expression also produces undefined results.

`)` The <right-parenthesis> shall be special when matched with a preceding <left-parenthesis>, both not inside a bracket expression.

`* + ? {` The <asterisk>, <plus-sign>, <question-mark>, and <left-brace> shall be special except when used in a bracket expression (see Section 9.3.5, on page 182). Any of the following uses produce undefined results:

- If these characters appear first in an ERE, or immediately following an unescaped <vertical-line>, <circumflex>, <dollar-sign>, or <left-parenthesis>
- If a <left-brace> is not part of a valid interval expression (see Section 9.4.6, on page 189)

`|` The <vertical-line> is special except when used in a bracket expression (see Section 9.3.5, on page 182). A <vertical-line> appearing first or last in an ERE, or immediately following a <vertical-line> or a <left-parenthesis>, or immediately preceding a <right-parenthesis>, produces undefined results.

`^` The <circumflex> shall be special when used as an anchor (see Section 9.4.9, on page 190). The <circumflex> shall signify a non-matching list expression when it occurs first in a list, immediately following a <left-square-bracket> (see Section 9.3.5, on page 182).

`$` The <dollar-sign> shall be special when used as an anchor.

9.4.4 Periods in EREs

When not inside a bracket expression, a <period> (`'.'`) is an ERE that shall match any character in the supported character set except NUL.

9.4.5 ERE Bracket Expression

The rules for ERE Bracket Expressions are the same as for Basic Regular Expressions; see Section 9.3.5 (on page 182).

9.4.6 EREs Matching Multiple Characters

The following rules shall be used to construct EREs matching multiple characters from EREs matching a single character:

1. A concatenation of EREs shall match the concatenation of the character sequences matched by each component of the ERE. A concatenation of EREs enclosed in parentheses shall match whatever the concatenation without the parentheses matches. For example, both the ERE "cd" and the ERE "(cd)" are matched by the third and fourth character of the string "abcdefabcdef".

2. When an ERE matching a single character or an ERE enclosed in parentheses is followed by the special character <plus-sign> ('+'), together with that <plus-sign> it shall match what one or more consecutive occurrences of the ERE would match. For example, the ERE "b+(bc)" matches the fourth to seventh characters in the string "acabbbcde". And, "[ab]+" and "[ab][ab]*" are equivalent.

3. When an ERE matching a single character or an ERE enclosed in parentheses is followed by the special character <asterisk> ('*'), together with that <asterisk> it shall match what zero or more consecutive occurrences of the ERE would match. For example, the ERE "b*c" matches the first character in the string "cabbbcde", and the ERE "b*cd" matches the third to seventh characters in the string "cabbbcdebbbbbbcdbc". And, "[ab]*" and "[ab][ab]" are equivalent when matching the string "ab".

4. When an ERE matching a single character or an ERE enclosed in parentheses is followed by the special character <question-mark> ('?'), together with that <question-mark> it shall match what zero or one consecutive occurrences of the ERE would match. For example, the ERE "b?c" matches the second character in the string "acabbbcde".

5. When an ERE matching a single character or an ERE enclosed in parentheses is followed by an interval expression of the format "{m}", "{m,}", or "{m,n}", together with that interval expression it shall match what repeated consecutive occurrences of the ERE would match. The values of *m* and *n* are decimal integers in the range 0 ≤*m*≤*n*≤{RE_DUP_MAX}, where *m* specifies the exact or minimum number of occurrences and *n* specifies the maximum number of occurrences. The expression "{m}" matches exactly *m* occurrences of the preceding ERE, "{m,}" matches at least *m* occurrences, and "{m,n}" matches any number of occurrences between *m* and *n*, inclusive.

   For example, in the string "abababccccccd" the ERE "c{3}" is matched by characters seven to nine and the ERE "(ab){2,}" is matched by characters one to six.

6. Each of the duplication symbols ('+', '*', '?', and intervals) can be suffixed by the repetition modifier '?' (<question-mark>), in which case matching behavior for that repetition shall be changed from the leftmost longest possible match to the leftmost shortest possible match, including the null match (see Section A.9, on page 3709). For example, the ERE ".*c" matches up to and including the last character ('c') in the string "abc abc", whereas the ERE ".*?c" matches up to and including the first character 'c', the third character in the string.

   If the REG_MINIMAL flag, defined in the **<regex.h>** header, is used when compiling an ERE via *regcomp*(), the leftmost shortest possible match shall be the default for all duplication symbols, and the repetition modifier '?' can be used to select the leftmost longest possible match for the repetition it modifies.

The behavior of multiple adjacent duplication symbols ('+', '*', '?', and intervals, possibly suffixed by the repetition modifier '?') produces undefined results.

An ERE matching a single character repeated by an '*', '?', or an interval expression shall not

match a null expression unless this is the only match for the repetition or it is necessary to satisfy the exact or minimum number of occurrences for the interval expression.

9.4.7 ERE Alternation

Two EREs separated by the special character <vertical-line> (`'|'`) shall match a string that is matched by either. For example, the ERE `"a((bc)|d)"` matches the string `"abc"` and the string `"ad"`. Single characters, or expressions matching single characters, separated by the <vertical-line> and enclosed in parentheses, shall be treated as an ERE matching a single character.

9.4.8 ERE Precedence

The order of precedence shall be as shown in the following table:

ERE Precedence (from high to low)	
Collation-related bracket symbols	`[==] [::] [..]`
Escaped characters	`\<special character>`
Bracket expression	`[]`
Grouping	`()`
Single-character-ERE duplication	`* + ? {m,n}`
Concatenation	
Anchoring	`^ $`
Alternation	`|`

For example, the ERE `"abba|cde"` matches either the string `"abba"` or the string `"cde"` (rather than the string `"abbade"` or `"abbcde"`, because concatenation has a higher order of precedence than alternation).

9.4.9 ERE Expression Anchoring

An ERE can be limited to matching expressions that begin or end a string; this is called ``anchoring''. The <circumflex> and <dollar-sign> special characters shall be considered ERE anchors when used anywhere except inside a bracket expression. This shall have the following effects:

1. When not inside a bracket expression, a <circumflex> (`'^'`) shall anchor the expression or subexpression it begins to the beginning of a string; such an expression or subexpression can match only a sequence starting at the first character of a string. For example, the EREs `"^ab"` and `"(^ab)"` match `"ab"` in the string `"abcdef"`, but fail to match in the string `"cdefab"`, and the ERE `"a^b"` is valid, but can never match because the `'a'` prevents the expression `"^b"` from matching starting at the first character.
2. When not inside a bracket expression, a <dollar-sign> (`'$'`) shall anchor the expression or subexpression it ends to the end of a string; such an expression or subexpression can match only a sequence ending at the last character of a string. For example, the EREs `"ef$"` and `"(ef$)"` match `"ef"` in the string `"abcdef"`, but fail to match in the string `"cdefab"`, and the ERE `"e$f"` is valid, but can never match because the `'f'` prevents the expression `"e$"` from matching ending at the last character.

9.5 Regular Expression Grammar

Grammars describing the syntax of both basic and extended regular expressions are presented in this section. The grammar takes precedence over the text. See XCU Section 1.3 (on page 2461).

9.5.1 BRE/ERE Grammar Lexical Conventions

The lexical conventions for regular expressions are as described in this section.

Except as noted, the longest possible token or delimiter beginning at a given point is recognized.

The following tokens are processed (in addition to those string constants shown in the grammar):

COLL_ELEM_SINGLE Any single-character collating element, unless it is a **META_CHAR**.

COLL_ELEM_MULTI Any multi-character collating element.

BACKREF Applicable only to basic regular expressions. The character string consisting of a <backslash> character followed by a single-digit numeral, `'1'` to `'9'`.

DUP_COUNT Represents a numeric constant. It shall be an integer in the range 0 ≤**DUP_COUNT** ≤{RE_DUP_MAX}. This token is only recognized when the context of the grammar requires it. At all other times, digits not preceded by a <backslash> character are treated as **ORD_CHAR**.

META_CHAR One of the characters:

- `^` When found first in a bracket expression
- – When found anywhere but first (after an initial `'^'`, if any) or last in a bracket expression, or as the ending range point in a range expression
- `]` When found anywhere but first (after an initial `'^'`, if any) in a bracket expression

L_ANCHOR Applicable only to basic regular expressions. The character `'^'` when it appears either as the first character of a basic regular expression or, if the implementation does not match the escape sequence `"\|"` to the literal character `'|'`, when used immediately following a `"\|"` escape sequence that is not inside a subexpression, and when not **QUOTED_CHAR**. The `'^'` may be recognized as an anchor elsewhere; see Section 9.3.8 (on page 186).

ORD_CHAR A character, other than one of the special characters in **SPEC_CHAR**.

QUOTED_CHAR In a BRE, one of the character sequences:

```
\^    \.    \*    \[    \]    \$    \\
```

On implementations where the escape sequences `"\?"`, `"\+"`, and `"\|"` match the literal characters `'?'`, `'+'`, and `'|'`, respectively, **QUOTED_CHAR** shall also include:

```
\?    \+    \|
```

In an ERE, one of the character sequences:

```
\^    \.    \[    \]    \$    \(    \)    \|
```


6852 `\*  \+  \?  \{  \}  \\`

6853 **R_ANCHOR** (Applicable only to basic regular expressions.) The character `'$'`
6854 when it appears either as the last character of a basic regular
6855 expression or, if the implementation does not match the escape
6856 sequence `"\|"` to the literal character `'|'`, when used immediately
6857 preceding a `"\|"` escape sequence that is not inside a subexpression,
6858 and when not **QUOTED_CHAR**. The `'$'` may be recognized as an
6859 anchor elsewhere; see Section 9.3.8 (on page 186).

6860 **SPEC_CHAR** For basic regular expressions, one of the following special characters:

6861 `.` Anywhere except inside bracket expressions

6862 `\` Anywhere except inside bracket expressions

6863 `[` Anywhere except inside bracket expressions

6864 `^` When used as an anchor (see Section 9.3.8, on page 186)

6865 `$` When used as an anchor

6866 `*` Anywhere except first in an entire RE, anywhere in a
6867 bracket expression, directly following `"\("`, directly
6868 following an anchoring `'^'`

6869 For extended regular expressions, shall be one of the following
6870 special characters found anywhere except inside bracket expressions:

6871 `^  .  [  $  (  )  |`
6872 `*  +  ?  {  \`

6873 The close-parenthesis shall be considered special in this context only
6874 if matched with a preceding open-parenthesis.

6875 ### 9.5.2 RE and Bracket Expression Grammar

6876 This section presents the grammar for basic regular expressions, including the bracket
6877 expression grammar that is common to both BREs and EREs.

```
6878 %token    ORD_CHAR QUOTED_CHAR DUP_COUNT
6879 %token    BACKREF L_ANCHOR R_ANCHOR
6880 %token    Back_open_paren  Back_close_paren
6881 /*          '\('             '\)'        */
6882 %token    Back_open_brace  Back_close_brace
6883 /*          '\{'             '\}'        */
6884 /* The following shall be tokens on implementations where
6885    \?, \+, and \| are not included in QUOTED_CHAR */
6886 %token    Back_qm  Back_plus  Back_bar
6887 /*          '\?'     '\+'       '\|'     */
6888 /* The following tokens are for the Bracket Expression
6889    grammar common to both REs and EREs. */
6890 %token    COLL_ELEM_SINGLE COLL_ELEM_MULTI META_CHAR
6891 %token    Open_equal Equal_close Open_dot Dot_close Open_colon Colon_close
```



```
6892        /*           '[='       '=]'        '[.'     '.]'      '[:'       ':]'  */
6893        %token    class_name
6894        /* class_name is a keyword to the LC_CTYPE locale category */
6895        /* (representing a character class) in the current locale */
6896        /* and is only recognized between [: and :] */
6897        %start    basic_reg_exp
6898        %%
6899        /* -----------------------------------------
6900           Basic Regular Expression
6901           -----------------------------------------
6902        */
6903        basic_reg_exp   :                          BRE_branch
6904                        | basic_reg_exp Back_bar BRE_branch /* if Back_bar
6905                                                               is a token */
6906                        ;
6907        BRE_branch      :            BRE_expression
6908                        | BRE_branch BRE_expression
6909                        ;
6910        BRE_expression  :          simple_BRE
6911                        | L_ANCHOR
6912                        |                      R_ANCHOR
6913                        | L_ANCHOR             R_ANCHOR
6914                        | L_ANCHOR simple_BRE
6915                        |          simple_BRE R_ANCHOR
6916                        | L_ANCHOR simple_BRE R_ANCHOR
6917                        ;
6918        simple_BRE      : nondupl_BRE
6919                        | nondupl_BRE BRE_dupl_symbol
6920                        ;
6921        nondupl_BRE     : one_char_or_coll_elem_BRE
6922                        | Back_open_paren basic_reg_exp Back_close_paren
6923                        | BACKREF
6924                        ;
6925        one_char_or_coll_elem_BRE  : ORD_CHAR
6926                        | QUOTED_CHAR
6927                        | '.'
6928                        | bracket_expression
6929                        ;
6930        BRE_dupl_symbol : '*'
6931                        | Back_qm   /* if Back_qm is a token */
6932                        | Back_plus /* if Back_plus is a token */
6933                        | Back_open_brace DUP_COUNT               Back_close_brace
6934                        | Back_open_brace DUP_COUNT ','           Back_close_brace
6935                        | Back_open_brace DUP_COUNT ',' DUP_COUNT Back_close_brace
6936                        ;
6937        /* -----------------------------------------
6938           Bracket Expression
6939           -----------------------------------------
6940        */
6941        bracket_expression : '[' matching_list ']'
6942                           | '[' nonmatching_list ']'
```



```
                ;
matching_list   : bracket_list
                ;
nonmatching_list : '^' bracket_list
                ;
bracket_list    : follow_list
                | follow_list '-'
                ;
follow_list     :             expression_term
                | follow_list expression_term
                ;
expression_term : single_expression
                | range_expression
                ;
single_expression : end_range
                | character_class
                | equivalence_class
                ;
range_expression : start_range end_range
                | start_range '-'
                ;
start_range     : end_range '-'
                ;
end_range       : COLL_ELEM_SINGLE
                | collating_symbol
                ;
collating_symbol : Open_dot COLL_ELEM_SINGLE Dot_close
                | Open_dot COLL_ELEM_MULTI Dot_close
                | Open_dot META_CHAR Dot_close
                ;
equivalence_class : Open_equal COLL_ELEM_SINGLE Equal_close
                | Open_equal COLL_ELEM_MULTI Equal_close
                ;
character_class : Open_colon class_name Colon_close
                ;
```

Note that although the BRE grammar appears always to permit **L_ANCHOR** or **R_ANCHOR** inside "\(" and "\)", the lexical conventions (see Section 9.5.1, on page 191) imply that '^' and '$' may be ordinary characters there. This reflects the semantic limits on the application, as noted in Section 9.3.8 (on page 186). Since it is an implementation option whether to interpret '^' and '$' as anchors in these locations, conforming applications cannot use unescaped '^' and '$' in positions inside "\(" and "\)" that might be interpreted as anchors.

9.5.3 ERE Grammar

This section presents the grammar for extended regular expressions, excluding the bracket expression grammar.

Note: The bracket expression grammar and the associated **%token** lines are identical between BREs and EREs. It has been omitted from the ERE section to avoid unnecessary editorial duplication.

```
%token  ORD_CHAR QUOTED_CHAR DUP_COUNT
%start  extended_reg_exp
%%
```



```
/* ------------------------------------------
   Extended Regular Expression
   ------------------------------------------
*/
extended_reg_exp   :                       ERE_branch
                   | extended_reg_exp '|' ERE_branch
                   ;
ERE_branch         :            ERE_expression
                   | ERE_branch ERE_expression
                   ;
ERE_expression     : one_char_or_coll_elem_ERE
                   | '^'
                   | '$'
                   | '(' extended_reg_exp ')'
                   | ERE_expression ERE_dupl_symbol
                   ;
one_char_or_coll_elem_ERE  : ORD_CHAR
                   | QUOTED_CHAR
                   | '.'
                   | bracket_expression
                   ;
ERE_dupl_symbol    : '*'
                   | '+'
                   | '?'
                   | '{' DUP_COUNT               '}'
                   | '{' DUP_COUNT ','           '}'
                   | '{' DUP_COUNT ',' DUP_COUNT '}'
                   ;
```

The ERE grammar does not permit several constructs that previous sections specify as having undefined results. Additionally, there are some constructs which the grammar permits but which still give undefined results:

- **ORD_CHAR** preceded by an unescaped <backslash> character
- One or more *ERE_dupl_symbol*s appearing first in an ERE, or immediately following `'|'`, `'^'`, `'('`, or `'$'`
- `'{'` not part of a valid *ERE_dupl_symbol*
- `'|'` appearing first or last in an ERE, or immediately following `'|'` or `'('`, or immediately preceding `')'`

Implementations are permitted to extend the language to allow these. Strictly Conforming applications cannot use such constructs.

Chapter 10
Directory Structure and Devices

10.1 Directory Structure and Files

The following directories shall exist on conforming systems and conforming applications shall make use of them only as described. Strictly conforming applications shall not assume the ability to create files in any of these directories, unless specified below.

/ The root directory.

/dev Contains **/dev/console**, **/dev/null**, and **/dev/tty**, described below.

The following directory shall exist on conforming systems and shall be used as described:

/tmp A directory made available for applications that need a place to create temporary files. Applications shall be allowed to create files in this directory, but shall not assume that such files are preserved between invocations of the application.

The following files shall exist on conforming systems and shall be both readable and writable:

/dev/null An empty data source and infinite data sink. Data written to **/dev/null** shall be discarded. Reads from **/dev/null** shall always return end-of-file (EOF).

/dev/tty In each process, a synonym for the controlling terminal associated with the process group of that process, if any. It is useful for programs or shell procedures that wish to be sure of writing messages to or reading data from the terminal no matter how output has been redirected. It can also be used for applications that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

The following file shall exist on conforming systems and need not be readable or writable:

/dev/console The **/dev/console** file is a generic name given to the system console (see Section 3.376, on page 86). It is usually linked to an implementation-defined special file. It shall provide an interface to the system console conforming to the requirements of Chapter 11 (on page 199).

10.2 Output Devices and Terminal Types

The utilities in the Shell and Utilities volume of POSIX.1-2024 historically have been implemented on a wide range of terminal types, but a conforming implementation need not support all features of all utilities on every conceivable terminal. POSIX.1-2024 states which features are optional for certain classes of terminals in the individual utility description sections. The implementation shall document in the system documentation which terminal types it supports and which of these features and utilities are not supported by each terminal.

When a feature or utility is not supported on a specific terminal type, as allowed by POSIX.1-2024, and the implementation considers such a condition to be an error preventing use of the feature or utility, the implementation shall indicate such conditions through diagnostic messages or exit status values or both (as appropriate to the specific utility description) that

inform the user that the terminal type lacks the appropriate capability.

POSIX.1-2024 uses a notational convention based on historical practice that identifies some of the control characters defined in Section 7.3.1 (on page 131) in a manner easily remembered by users on many terminals. The correspondence between this ``<control>-*char*'' notation and the actual control characters is shown in the following table. When POSIX.1-2024 refers to a character by its <control>-*name*, it is referring to the actual control character shown in the Value column of the table, which is not necessarily the exact control key sequence on all terminals. Some terminals have keyboards that do not allow the direct transmission of all the non-alphanumeric characters shown. In such cases, the system documentation shall describe which data sequences transmitted by the terminal are interpreted by the system as representing the special characters.

Table 10-1 Control Character Names

Name	Value	Name	Value
<control>-A	<SOH>	<control>-Q	<DC1>
<control>-B	<STX>	<control>-R	<DC2>
<control>-C	<ETX>	<control>-S	<DC3>
<control>-D	<EOT>	<control>-T	<DC4>
<control>-E	<ENQ>	<control>-U	<NAK>
<control>-F	<ACK>	<control>-V	<SYN>
<control>-G	<BEL>	<control>-W	<ETB>
<control>-H	<BS>	<control>-X	<CAN>
<control>-I	<HT>	<control>-Y	<EM>
<control>-J	<LF>	<control>-Z	<SUB>
<control>-K	<VT>	<control>-[	<ESC>
<control>-L	<FF>	<control>-\	<FS>
<control>-M	<CR>	<control>-]	<GS>
<control>-N	<SO>	<control>-^	<RS>
<control>-O	<SI>	<control>-_	<US>
<control>-P	<DLE>	<control>-?	<DEL>

Note: The notation uses uppercase letters for arbitrary editorial reasons. There is no implication that the keystrokes represent control-shift-letter sequences.

Chapter 11

General Terminal Interface

This chapter describes a general terminal interface that shall be provided. It shall be supported on any asynchronous communications ports if the implementation provides them. It is implementation-defined whether it supports network connections or synchronous ports, or both.

11.1 Interface Characteristics

11.1.1 Opening a Terminal Device File

When a terminal device file is opened, it normally causes the thread to wait until a connection is established. In practice, application programs seldom open these files; they are opened by special programs and become an application's standard input, output, and error files.

Cases where applications do open a terminal device are as follows:

1. Opening **/dev/tty**, or the pathname returned by *ctermid*( ), in order to obtain a file descriptor for the controlling terminal; see Section 11.1.3 (on page 200).
2. Opening the subsidiary side of a pseudo-terminal; see XSH *ptsname*( ).
3. Opening a modem or similar piece of equipment connected by a serial line. In this case, the terminal parameters (see Section 11.2, on page 205) may be initialized to default settings by the implementation in between the last close of the device by any process and the next open of the device, or they may persist from one use to the next. The terminal parameters can be set to values that ensure the terminal behaves in a conforming manner by means of the O_TTY_INIT open flag when opening a terminal device that is not already open in any process, or by executing the *stty* utility with the operand **sane**.

As described in *open*( ), opening a terminal device file with the O_NONBLOCK flag clear shall cause the thread to block until the terminal device is ready and available. If CLOCAL mode is not set, this means blocking until a connection is established. If CLOCAL mode is set in the terminal, or the O_NONBLOCK flag is specified in the *open*( ), the *open*( ) function shall return a file descriptor without waiting for a connection to be established.

11.1.2 Process Groups

A terminal may have a foreground process group associated with it. This foreground process group plays a special role in handling signal-generating input characters, as discussed in Section 11.1.9 (on page 203).

A command interpreter process supporting job control can allocate the terminal to different jobs, or process groups, by placing related processes in a single process group and associating this process group with the terminal. A terminal's foreground process group may be set or examined by a process, assuming the permission requirements are met; see *tcgetpgrp*( ) and *tcsetpgrp*( ).

The terminal interface aids in this allocation by restricting access to the terminal by processes that are not in the current process group; see Section 11.1.4.

When there is no longer any process whose process ID or process group ID matches the foreground process group ID, the terminal shall have no foreground process group. It is unspecified whether the terminal has a foreground process group when there is a process whose process ID matches the foreground process group ID, but whose process group ID does not. No actions defined in POSIX.1-2024, other than allocation of a controlling terminal or a successful call to *tcsetpgrp*(), shall cause a process group to become the foreground process group of the terminal.

11.1.3 The Controlling Terminal

A terminal may belong to a process as its controlling terminal. Each process of a session that has a controlling terminal has the same controlling terminal. A terminal may be the controlling terminal for at most one session. The controlling terminal for a session is allocated by the session leader in an implementation-defined manner. If a session leader has no controlling terminal, and opens a terminal device file that is not already associated with a session without using the O_NOCTTY option (see *open*()), it is implementation-defined whether the terminal becomes the controlling terminal of the session leader. If a process which is not a session leader opens a terminal file, or the O_NOCTTY option is used on *open*(), then that terminal shall not become the controlling terminal of the calling process. When a controlling terminal becomes associated with a session, its foreground process group shall be set to the process group of the session leader.

The controlling terminal is inherited by a child process during a *fork*() function call. A process relinquishes its controlling terminal when it creates a new session with the *setsid*() function; other processes remaining in the old session that had this terminal as their controlling terminal continue to have it. Upon the close of the last file descriptor in the system (whether or not it is in the current session) associated with the controlling terminal, it is unspecified whether all processes that had that terminal as their controlling terminal cease to have any controlling terminal. Whether and how a session leader can reacquire a controlling terminal after the controlling terminal has been relinquished in this fashion is unspecified. A process does not relinquish its controlling terminal simply by closing all of its file descriptors associated with the controlling terminal if other processes continue to have it open.

When a controlling process terminates, the controlling terminal is dissociated from the current session, allowing it to be acquired by a new session leader. Subsequent access to the terminal by other processes in the earlier session may be denied, with attempts to access the terminal treated as if a modem disconnect had been sensed.

11.1.4 Terminal Access Control

If a process is in the foreground process group of its controlling terminal, read operations shall be allowed, as described in Section 11.1.5 (on page 201). Any attempts by a process in a background process group to read from its controlling terminal cause its process group to be sent a SIGTTIN signal unless one of the following special cases applies: if the reading process is ignoring the SIGTTIN signal or the reading thread is blocking the SIGTTIN signal, or if the process group of the reading process is orphaned, the *read*() shall return –1, with *errno* set to [EIO] and no signal shall be sent. The default action of the SIGTTIN signal shall be to stop the process to which it is sent. See **<signal.h>**.

If a process is in the foreground process group of its controlling terminal, write operations shall

be allowed as described in Section 11.1.8 (on page 203). Attempts by a process in a background process group to write to its controlling terminal shall cause the process group to be sent a SIGTTOU signal unless one of the following special cases applies: if TOSTOP is not set, or if TOSTOP is set and the process is ignoring the SIGTTOU signal or the writing thread is blocking the SIGTTOU signal, the process is allowed to write to the terminal and the SIGTTOU signal is not sent. If TOSTOP is set, the process group of the writing process is orphaned, the writing process is not ignoring the SIGTTOU signal, and the writing thread is not blocking the SIGTTOU signal, the *write*( ) shall return –1, with *errno* set to [EIO] and no signal shall be sent.

Certain calls that set terminal parameters are treated in the same fashion as *write*( ), except that TOSTOP is ignored; that is, the effect is identical to that of terminal writes when TOSTOP is set (see Section 11.2.5 (on page 210), *tcdrain*( ), *tcflow*( ), *tcflush*( ), *tcsendbreak*( ), *tcsetattr*( ), *tcsetpgrp*( ), and *tcsetwinsize*( )).

11.1.5 Input Processing and Reading Data

A terminal device associated with a terminal device file may operate in full-duplex mode, so that data may arrive even while output is occurring. Each terminal device file has an input queue associated with it, into which incoming data is stored by the system before being read by a process. The system may impose a limit, {MAX_INPUT}, on the number of bytes that may be stored in the input queue. The behavior of the system when this limit is exceeded is implementation-defined.

Two general kinds of input processing are available, determined by whether the terminal device file is in canonical mode or non-canonical mode. These modes are described in Section 11.1.6 (on page 202) and Section 11.1.7 (on page 202). Additionally, input characters are processed according to the *c_iflag* (see Section 11.2.2, on page 206) and *c_lflag* (see Section 11.2.5, on page 210) fields. Such processing can include ``echoing'', which in general means transmitting input characters immediately back to the terminal when they are received from the terminal. This is useful for terminals that can operate in full-duplex mode.

The manner in which data is provided to a process reading from a terminal device file is dependent on whether the terminal file is in canonical or non-canonical mode, and on whether or not the O_NONBLOCK flag is set by *open*( ) or *fcntl*( ).

If the O_NONBLOCK flag is clear, then the read request shall be blocked until data is available or a signal has been received. If the O_NONBLOCK flag is set, then the read request shall be completed, without blocking, in one of three ways:

1. If there is enough data available to satisfy the entire request, the *read*( ) shall complete successfully and shall return the number of bytes read.
2. If there is not enough data available to satisfy the entire request, the *read*( ) shall complete successfully, having read as much data as possible, and shall return the number of bytes it was able to read.
3. If there is no data available, the *read*( ) shall return –1, with *errno* set to [EAGAIN].

When data is available depends on whether the input processing mode is canonical or non-canonical. Section 11.1.6 (on page 202) and Section 11.1.7 (on page 202) describe each of these input processing modes.

11.1.6 Canonical Mode Input Processing

In canonical mode input processing, terminal input is processed in units of lines. A line is delimited by a <newline> character (NL), an end-of-file character (EOF), or an end-of-line (EOL) character. See Section 11.1.9 (on page 203) for more information on EOF and EOL. This means that a read request shall not return until an entire line has been typed or a signal has been received. Also, no matter how many bytes are requested in the *read*( ) call, at most one line shall be returned. It is not, however, necessary to read a whole line at once; any number of bytes, even one, may be requested in a *read*( ) without losing information.

If {MAX_CANON} is defined for this terminal device, it shall be a limit on the number of bytes in a line. The behavior of the system when this limit is exceeded is implementation-defined. If {MAX_CANON} is not defined, there shall be no such limit; see *pathconf*( ).

Erase and kill processing occur when either of two special characters, the ERASE and KILL characters (see Section 11.1.9, on page 203), is received. This processing shall affect data in the input queue that has not yet been delimited by an NL, EOF, or EOL character. This un-delimited data makes up the current line. The ERASE character shall delete the last character in the current line, if there is one. The KILL character shall delete all data in the current line, if there is any. The ERASE and KILL characters shall have no effect if there is no data in the current line. The ERASE and KILL characters themselves shall not be placed in the input queue.

11.1.7 Non-Canonical Mode Input Processing

In non-canonical mode input processing, input bytes are not assembled into lines, and erase and kill processing shall not occur. The values of the MIN and TIME members of the *c_cc* array are used to determine how to process the bytes received. POSIX.1-2024 does not specify whether the setting of O_NONBLOCK takes precedence over MIN or TIME settings. Therefore, if O_NONBLOCK is set, *read*( ) may return immediately, regardless of the setting of MIN or TIME. Also, if no data is available, *read*( ) may either return 0, or return –1 with *errno* set to [EAGAIN].

MIN represents the minimum number of bytes that should be received when the *read*( ) function returns successfully. TIME is a timer of 0.1 second granularity that is used to time out bursty and short-term data transmissions. If MIN is greater than {MAX_INPUT}, the response to the request is undefined. The four possible values for MIN and TIME and their interactions are described below.

Case A: MIN>0, TIME>0

In case A, TIME serves as an inter-byte timer which shall be activated after the first byte is received. Since it is an inter-byte timer, it shall be reset after a byte is received. The interaction between MIN and TIME is as follows. As soon as one byte is received, the inter-byte timer shall be started. If MIN bytes are received before the inter-byte timer expires (remember that the timer is reset upon receipt of each byte), the read shall be satisfied. If the timer expires before MIN bytes are received, the characters received to that point shall be returned to the user. Note that if TIME expires at least one byte shall be returned because the timer would not have been enabled unless a byte was received. In this case (MIN>0, TIME>0) the read shall block until the MIN and TIME mechanisms are activated by the receipt of the first byte, or a signal is received. If data is in the buffer at the time of the *read*( ), the result shall be as if data has been received immediately after the *read*( ).

Case B: MIN>0, TIME=0

In case B, since the value of TIME is zero, the timer plays no role and only MIN is significant. A pending read shall not be satisfied until MIN bytes are received (that is, the pending read shall block until MIN bytes are received), or a signal is received. A program that uses case B to read record-based terminal I/O may block indefinitely in the read operation.

Case C: MIN=0, TIME>0

In case C, since MIN=0, TIME no longer represents an inter-byte timer. It now serves as a read timer that shall be activated as soon as the *read*( ) function is processed. A read shall be satisfied as soon as a single byte is received or the read timer expires. Note that in case C if the timer expires, no bytes shall be returned. If the timer does not expire, the only way the read can be satisfied is if a byte is received. If bytes are not received, the read shall not block indefinitely waiting for a byte; if no byte is received within TIME*0.1 seconds after the read is initiated, the *read*( ) shall return a value of zero, having read no data. If data is in the buffer at the time of the *read*( ), the timer shall be started as if data has been received immediately after the *read*( ).

Case D: MIN=0, TIME=0

The minimum of either the number of bytes requested or the number of bytes currently available shall be returned without waiting for more bytes to be input. If no characters are available, *read*( ) shall return a value of zero, having read no data.

11.1.8 Writing Data and Output Processing

When a process writes one or more bytes to a terminal device file, they are processed according to the *c_oflag* field (see Section 11.2.3, on page 207). The implementation may provide a buffering mechanism; as such, when a call to *write*( ) completes, all of the bytes written have been scheduled for transmission to the device, but the transmission has not necessarily completed. See *write*( ) for the effects of O_NONBLOCK on *write*( ).

11.1.9 Special Characters

Certain characters have special functions on input or output or both. These functions are summarized as follows:

INTR Special character on input, which is recognized if the ISIG flag is set. Generates a SIGINT signal which is sent to all processes in the foreground process group for which the terminal is the controlling terminal. If ISIG is set, the INTR character shall be discarded when processed.

QUIT Special character on input, which is recognized if the ISIG flag is set. Generates a SIGQUIT signal which is sent to all processes in the foreground process group for which the terminal is the controlling terminal. If ISIG is set, the QUIT character shall be discarded when processed.

ERASE Special character on input, which is recognized if the ICANON flag is set. Erases the last character in the current line; see Section 11.1.6 (on page 202). It shall not erase beyond the start of a line, as delimited by an NL, EOF, or EOL character. If ICANON is set, the ERASE character shall be discarded when processed.

KILL Special character on input, which is recognized if the ICANON flag is set. Deletes the entire line, as delimited by an NL, EOF, or EOL character. If ICANON is set, the KILL character shall be discarded when processed.

EOF Special character on input, which is recognized if the ICANON flag is set. When received, all the bytes waiting to be read are immediately passed to the process without waiting for a <newline>, and the EOF is discarded. Thus, if there are no bytes waiting (that is, the EOF occurred at the beginning of a line), a byte count of zero shall be returned from the *read*( ), representing an end-of-file indication. If ICANON is set, the EOF character shall be discarded when processed.

NL Special character on input, which is recognized if the ICANON flag is set. It is the line delimiter <newline>. It cannot be changed.

EOL Special character on input, which is recognized if the ICANON flag is set. It is an additional line delimiter, like NL.

SUSP If the ISIG flag is set, receipt of the SUSP character shall cause a SIGTSTP signal to be sent to all processes in the foreground process group for which the terminal is the controlling terminal, and the SUSP character shall be discarded when processed.

STOP Special character on both input and output, which is recognized if the IXON (output control) or IXOFF (input control) flag is set. Can be used to suspend output temporarily. It is useful with CRT terminals to prevent output from disappearing before it can be read. If IXON is set, the STOP character shall be discarded when processed.

START Special character on both input and output, which is recognized if the IXON (output control) or IXOFF (input control) flag is set. Can be used to resume output that has been suspended by a STOP character. If IXON is set, the START character shall be discarded when processed.

CR Special character on input, which is recognized if the ICANON flag is set; it is the <carriage-return> character. When ICANON and ICRNL are set and IGNCR is not set, this character shall be translated into an NL, and shall have the same effect as an NL character. It cannot be changed.

The NL and CR characters cannot be changed. It is implementation-defined whether the START and STOP characters can be changed. The values for INTR, QUIT, ERASE, KILL, EOF, EOL, and SUSP shall be changeable to suit individual tastes. Special character functions associated with changeable special control characters can be disabled individually.

If two or more special characters have the same value, the function performed when that character is received is undefined.

A special character is recognized not only by its value, but also by its context; for example, an implementation may support multi-byte sequences that have a meaning different from the meaning of the bytes when considered individually. Implementations may also support additional single-byte functions. These implementation-defined multi-byte or single-byte functions shall be recognized only if the IEXTEN flag is set; otherwise, data is received without interpretation, except as required to recognize the special characters defined in this section.

XSI If IEXTEN is set, the ERASE, KILL, and EOF characters can be escaped by a preceding <backslash> character, in which case no special function shall occur.

11.1.10 Modem Disconnect

If a modem disconnect is detected by the terminal interface for a controlling terminal, and if CLOCAL is not set in the *c_cflag* field for the terminal (see Section 11.2.4, on page 209), the SIGHUP signal shall be sent to the controlling process for which the terminal is the controlling terminal. Unless other arrangements have been made, this shall cause the controlling process to terminate (see *exit*( )). Any subsequent read from the terminal device shall return the value of zero, indicating end-of-file; see *read*( ). Thus, processes that read a terminal file and test for end-of-file can terminate appropriately after a disconnect. If the EIO condition as specified in *read*( ) also exists, it is unspecified whether on EOF condition or [EIO] is returned. Any subsequent *write*( ) to the terminal device shall return –1, with *errno* set to [EIO], until the device is closed.

11.1.11 Closing a Terminal Device File

The last process to close a terminal device file shall cause any output to be sent to the device and shall cause any input to be discarded. If HUPCL is set in the control modes and the communications port supports a disconnect function, the terminal device shall perform a disconnect.

11.2 Parameters that Can be Set

11.2.1 The termios Structure

Routines that need to control certain terminal I/O characteristics shall do so by using the **termios** structure as defined in the **<termios.h>** header.

Since the **termios** structure may include additional members, and the standard members may include both standard and non-standard modes, the structure should never be initialized directly by the application as this may cause the terminal to behave in a non-conforming manner. When opening a terminal device (other than a pseudo-terminal) that is not already open in any process, it should be opened with the O_TTY_INIT flag before initializing the structure using *tcgetattr*( ) to ensure that any non-standard elements of the **termios** structure are set to values that result in conforming behavior of the terminal interface.

The members of the **termios** structure include (but are not limited to):

Member Type	Array Size	Member Name	Description
tcflag_t		*c_iflag*	Input modes.
tcflag_t		*c_oflag*	Output modes.
tcflag_t		*c_cflag*	Control modes.
tcflag_t		*c_lflag*	Local modes.
cc_t	NCCS	*c_cc*[ ]	Control characters.

The **tcflag_t** and **cc_t** types are defined in the **<termios.h>** header. They shall be unsigned integer types.

11.2.2 Input Modes

Values of the *c_iflag* field describe the basic terminal input control, and are composed of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in this table are defined in **<termios.h>**:

Mask Name	Description
BRKINT	Signal interrupt on break.
ICRNL	Map CR to NL on input.
IGNBRK	Ignore break condition.
IGNCR	Ignore CR.
IGNPAR	Ignore characters with parity errors.
INLCR	Map NL to CR on input.
INPCK	Enable input parity check.
ISTRIP	Strip character.
IXANY	Enable any character to restart output.
IXOFF	Enable start/stop input control.
IXON	Enable start/stop output control.
PARMRK	Mark parity errors.

In the context of asynchronous serial data transmission, a break condition shall be defined as a sequence of zero-valued bits that continues for more than the time to send one byte. The entire sequence of zero-valued bits is interpreted as a single break condition, even if it continues for a time equivalent to more than one byte. In contexts other than asynchronous serial data transmission, the definition of a break condition is implementation-defined.

If IGNBRK is set, a break condition detected on input shall be ignored; that is, not put on the input queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the break condition shall flush the input and output queues, and if the terminal is the controlling terminal of a foreground process group, the break condition shall generate a single SIGINT signal to that foreground process group. If neither IGNBRK nor BRKINT is set, a break condition shall be read as a single 0x00, or if PARMRK is set, as 0xff 0x00 0x00.

If IGNPAR is set, a byte with a framing or parity error (other than break) shall be ignored.

If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than break) shall be given to the application as the three-byte sequence 0xff 0x00 X, where 0xff 0x00 is a two-byte flag preceding each sequence and X is the data of the byte received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid byte of 0xff is given to the application as 0xff 0xff. If neither PARMRK nor IGNPAR is set, a framing or parity error (other than break) shall be given to the application as a single byte 0x00.

If INPCK is set, input parity checking shall be enabled. If INPCK is not set, input parity checking shall be disabled, allowing output parity generation without input parity errors. Note that whether input parity checking is enabled or disabled is independent of whether parity detection is enabled or disabled (see Section 11.2.4, on page 209). If parity detection is enabled but input parity checking is disabled, the hardware to which the terminal is connected shall recognize the parity bit, but the terminal special file shall not check whether or not this bit is correctly set.

If ISTRIP is set, valid input bytes shall first be stripped to seven bits; otherwise, all eight bits shall be processed.

If INLCR is set, a received NL character shall be translated into a CR character. If IGNCR is set, a received CR character shall be ignored (not read). If IGNCR is not set and ICRNL is set, a received CR character shall be translated into an NL character.

If IXANY is set, any input character shall restart output that has been suspended.

If IXON is set, start/stop output control shall be enabled. A received STOP character shall suspend output and a received START character shall restart output. When IXON is set, START and STOP characters are not read, but merely perform flow control functions. When IXON is not set, the START and STOP characters shall be read.

If IXOFF is set, start/stop input control shall be enabled. The system shall transmit STOP characters, which are intended to cause the terminal device to stop transmitting data, as needed to prevent the input queue from overflowing and causing implementation-defined behavior, and shall transmit START characters, which are intended to cause the terminal device to resume transmitting data, as soon as the device can continue transmitting data without risk of overflowing the input queue. The precise conditions under which STOP and START characters are transmitted are implementation-defined.

The initial input control value after *open*( ) is implementation-defined.

11.2.3 Output Modes

The *c_oflag* field specifies the terminal interface's treatment of output, and is composed of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in the following table are defined in **<termios.h>**:

Mask Name	Description
OPOST	Perform output processing.
ONLCR	Map NL to CR-NL on output.
OCRNL	Map CR to NL on output.
ONOCR	No CR output at column 0.
ONLRET	NL performs CR function.
OFILL	Use fill characters for delay.
OFDEL	Fill is DEL, else NUL.
NLDLY	Select newline delays:
NL0	Newline character type 0.
NL1	Newline character type 1.
CRDLY	Select carriage-return delays:
CR0	Carriage-return delay type 0.
CR1	Carriage-return delay type 1.
CR2	Carriage-return delay type 2.
CR3	Carriage-return delay type 3.
TABDLY	Select horizontal-tab delays:
TAB0	Horizontal-tab delay type 0.
TAB1	Horizontal-tab delay type 1.
TAB2	Horizontal-tab delay type 2.
TAB3	Expand tabs to spaces.
BSDLY	Select backspace delays:
BS0	Backspace-delay type 0.
BS1	Backspace-delay type 1.
VTDLY	Select vertical-tab delays:
VT0	Vertical-tab delay type 0.
VT1	Vertical-tab delay type 1.
FFDLY	Select form-feed delays:
FF0	Form-feed delay type 0.
FF1	Form-feed delay type 1.

If OPOST is set, output data shall be post-processed as described below, so that lines of text are modified to appear appropriately on the terminal device; otherwise, characters shall be transmitted without change.

XSI If ONLCR is set, the NL character shall be transmitted as the CR-NL character pair. If OCRNL is set, the CR character shall be transmitted as the NL character. If ONOCR is set, no CR character shall be transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer shall be set to 0 and the delays specified for CR shall be used. Otherwise, the NL character is assumed to do just the line-feed function; the column pointer remains unchanged. The column pointer shall also be set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 shall indicate no delay. If OFILL is set, fill characters shall be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character shall be DEL; otherwise, NUL.

If a <form-feed> or <vertical-tab> delay is specified, it shall last for about 2 seconds.

Newline delay shall last about 0.10 seconds. If ONLRET is set, the carriage-return delays shall be used instead of the newline delays. If OFILL is set, two fill characters shall be transmitted.

Carriage-return delay type 1 shall be dependent on the current column position, type 2 shall be

about 0.10 seconds, and type 3 shall be about 0.15 seconds. If OFILL is set, delay type 1 shall transmit two fill characters, and type 2 four fill characters.

Horizontal-tab delay type 1 shall be dependent on the current column position. Type 2 shall be about 0.10 seconds. Type 3 specifies that <tab> characters shall be expanded into <space> characters. If OFILL is set, two fill characters shall be transmitted for any delay.

Backspace delay shall last about 0.05 seconds. If OFILL is set, one fill character shall be transmitted.

The actual delays depend on line speed and system load.

The initial output control value after *open*( ) is implementation-defined.

11.2.4 Control Modes

The *c_cflag* field describes the hardware control of the terminal, and is composed of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in this table are defined in **<termios.h>**; not all values specified are required to be supported by the underlying hardware (if any). If the terminal is a pseudo-terminal, it is unspecified whether non-default values are unsupported, or are supported and emulated in software, or are handled by *tcsetattr*( ), *tcgetattr*( ), and the *stty* utility as if they are supported but have no effect on the behavior of the terminal interface.

Mask Name	Description
CLOCAL	Ignore modem status lines.
CREAD	Enable receiver.
CSIZE	Number of bits transmitted or received per byte:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits.
CSTOPB	Send two stop bits, else one.
HUPCL	Hang up on last close.
PARENB	Parity enable.
PARODD	Odd parity, else even.

In addition, the input and output baud rates are stored in the **termios** structure. The symbols in the following table are defined in **<termios.h>**. Not all values specified are required to be supported by the underlying hardware (if any). For pseudo-terminals, the input and output baud rates set in the **termios** structure need not affect the speed of data transmission through the terminal interface.

Note: The term ``baud'' is used historically here, but is not technically correct. This is properly ``bits per second'', which may not be the same as baud. However, the term is used because of the historical usage and understanding.

Name	Description	Name	Description
B0	Hang up	B600	600 baud
B50	50 baud	B1200	1200 baud
B75	75 baud	B1800	1800 baud
B110	110 baud	B2400	2400 baud
B134	134.5 baud	B4800	4800 baud
B150	150 baud	B9600	9600 baud
B200	200 baud	B19200	19200 baud
B300	300 baud	B38400	38400 baud

The following functions are provided for getting and setting the values of the input and output baud rates in the **termios** structure: *cfgetispeed*( ), *cfgetospeed*( ), *cfsetispeed*( ), and *cfsetospeed*( ). The effects on the terminal device shall not become effective and not all errors need be detected until the *tcsetattr*( ) function is successfully called.

The CSIZE bits shall specify the number of transmitted or received bits per byte. If ISTRIP is not set, the value of all the other bits is unspecified. If ISTRIP is set, the value of all but the 7 low-order bits shall be zero, but the value of any other bits beyond CSIZE is unspecified when read. CSIZE shall not include the parity bit, if any. If CSTOPB is set, two stop bits shall be used; otherwise, one stop bit. For example, at 110 baud, two stop bits are normally used.

If CREAD is set, the receiver shall be enabled; otherwise, no characters shall be received.

If PARENB is set, parity generation and detection shall be enabled and a parity bit is added to each byte. If parity is enabled, PARODD shall specify odd parity if set; otherwise, even parity shall be used.

If HUPCL is set, the modem control lines for the port shall be lowered when the last process with the port open closes the port or the process terminates. The modem connection shall be broken.

If CLOCAL is set, a connection shall not depend on the state of the modem status lines. If CLOCAL is clear, the modem status lines shall be monitored.

Under normal circumstances, a call to the *open*( ) function shall wait for the modem connection to complete. However, if the O_NONBLOCK flag is set (see *open*( )) or if CLOCAL has been set, the *open*( ) function shall return immediately without waiting for the connection.

If the object for which the control modes are set is not an asynchronous serial connection, some of the modes may be ignored; for example, if an attempt is made to set the baud rate on a network connection to a terminal on another host, the baud rate need not be set on the connection between that terminal and the machine to which it is directly connected.

The initial hardware control value after *open*( ) is implementation-defined.

11.2.5 Local Modes

The *c_lflag* field of the argument structure is used to control various functions. It is composed of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in this table are defined in **<termios.h>**.

Mask Name	Description
ECHO	Enable echo.
ECHOE	Echo ERASE as an error correcting backspace.
ECHOK	Echo KILL.
ECHONL	Echo <newline>.
ICANON	Canonical input (erase and kill processing).
IEXTEN	Enable extended (implementation-defined) functions.
ISIG	Enable signals.
NOFLSH	Disable flush after interrupt, quit, or suspend.
TOSTOP	Send SIGTTOU for background output.

If ECHO is set, input characters shall be echoed back to the terminal. If ECHO is clear, input characters shall not be echoed.

If ECHOE and ICANON are set, the ERASE character shall cause the terminal to erase, if possible, the last character in the current line from the display. If there is no character to erase, an implementation may echo an indication that this was the case, or do nothing.

If ECHOK and ICANON are set, the KILL character shall either cause the terminal to erase the line from the display or shall echo the <newline> character after the KILL character.

If ECHONL and ICANON are set, the <newline> character shall be echoed even if ECHO is not set.

If ICANON is set, canonical processing shall be enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL, as described in Section 11.1.6 (on page 202).

If ICANON is not set, read requests shall be satisfied directly from the input queue. A read shall not be satisfied until at least MIN bytes have been received or the timeout value TIME expired between bytes. The time value represents tenths of a second. See Section 11.1.7 (on page 202) for more details.

If IEXTEN is set, implementation-defined functions shall be recognized from the input data. It is implementation-defined how IEXTEN being set interacts with ICANON, ISIG, IXON, or IXOFF. If IEXTEN is not set, implementation-defined functions shall not be recognized and the corresponding input characters are processed as described for ICANON, ISIG, IXON, and IXOFF.

If ISIG is set, each input character shall be checked against the special control characters INTR, QUIT, and SUSP. If an input character matches one of these control characters, the function associated with that character shall be performed. If ISIG is not set, no checking shall be done. Thus these special input functions are possible only if ISIG is set.

If NOFLSH is set, the normal flush of the input and output queues associated with the INTR, QUIT, and SUSP characters shall not be done.

If TOSTOP is set, the signal SIGTTOU shall be sent to the process group of a process that tries to write to its controlling terminal if it is not in the foreground process group for that terminal. This signal, by default, stops the members of the process group. Otherwise, the output generated by that process shall be output to the current output stream. If the writing process is ignoring the SIGTTOU signal or the writing thread is blocking the SIGTTOU signal, the process is allowed to produce output, and the SIGTTOU signal shall not be sent.

The initial local control value after *open*( ) is implementation-defined.

11.2.6 Special Control Characters

The special control character values shall be defined by the array *c_cc*. The subscript name and description for each element in both canonical and non-canonical modes are as follows:

Subscript Usage		
Canonical Mode	**Non-Canonical Mode**	**Description**
VEOF		EOF character
VEOL		EOL character
VERASE		ERASE character
VINTR	VINTR	INTR character
VKILL		KILL character
	VMIN	MIN value
VQUIT	VQUIT	QUIT character
VSUSP	VSUSP	SUSP character
	VTIME	TIME value
VSTART	VSTART	START character
VSTOP	VSTOP	STOP character

The subscript values are unique, except that the VMIN and VTIME subscripts may have the same values as the VEOF and VEOL subscripts, respectively.

Implementations that do not support changing the START and STOP characters may ignore the character values in the *c_cc* array indexed by the VSTART and VSTOP subscripts when *tcsetattr*( ) is called, but shall return the value in use when *tcgetattr*( ) is called.

The initial values of all control characters are implementation-defined.

If the value of one of the changeable special control characters (see Section 11.1.9, on page 203) is _POSIX_VDISABLE, that function shall be disabled; that is, no input data is recognized as the disabled special character. If ICANON is not set, the value of _POSIX_VDISABLE has no special meaning for the VMIN and VTIME entries of the *c_cc* array.

Chapter 12

Utility Conventions

12.1 Utility Argument Syntax

This section describes the argument syntax of the standard utilities and introduces terminology used throughout POSIX.1-2024 for describing the arguments processed by the utilities.

Within POSIX.1-2024, a special notation is used for describing the syntax of a utility's arguments. Unless otherwise noted, all utility descriptions use this notation, which is illustrated by this example (see XCU Section 2.9.1, on page 2500):

```
utility_name [-a] [-b] [-c option_argument]
    [-d|-e] [-f[option_argument]] [operand...]
```

The notation used for the SYNOPSIS sections imposes requirements on the implementors of the standard utilities and provides a simple reference for the application developer or system user.

1. The utility in the example is named *utility_name*. It is followed by options, option-arguments, and operands. The arguments that consist of <hyphen-minus> characters and single letters or digits, such as `'a'`, are known as ``options'' (or, historically, ``flags''). Certain options are followed by an ``option-argument'', as shown with [**–c** *option_argument*]. The arguments following the last options and option-arguments are named ``operands''.

2. Option-arguments are shown separated from their options by <blank> characters, except when the option-argument is enclosed in the `'['` and `']'` notation to indicate that it is optional. This reflects the situation in which an optional option-argument (if present) is included within the same argument string as the option; for a mandatory option-argument, it is the next argument. The Utility Syntax Guidelines in Section 12.2 (on page 215) require that the option be a separate argument from its option-argument and that option-arguments not be optional, but there are some exceptions in POSIX.1-2024 to provide for continued operation of historical applications:

    a. If the SYNOPSIS of a standard utility shows an option with a mandatory option-argument (as with [**–c** *option_argument*] in the example), a conforming application shall use separate arguments for that option and its option-argument. However, a conforming implementation shall also permit applications to specify the option and option-argument in the same argument string without intervening <blank> characters.

    b. If the SYNOPSIS shows an optional option-argument (as with [**–f**[*option_argument*]] in the example), a conforming application shall place any option-argument for that option directly adjacent to the option in the same argument string, without intervening <blank> characters. If the utility receives an argument containing only the option, it shall behave as specified in its description for an omitted option-argument; it shall not treat the next argument (if any) as the option-argument for that option.

3. Options are usually listed in alphabetical order unless this would make the utility description more confusing. There are no implied relationships between the options based upon the order in which they appear, unless otherwise stated in the OPTIONS section, or unless the exception in Guideline 11 of Section 12.2 (on page 215) applies. If an option that does not have option-arguments is repeated, the results are undefined, unless otherwise stated.

4. Frequently, names of parameters that require substitution by actual values are shown with embedded <underscore> characters. Alternatively, parameters are shown as follows:

   `<parameter name>`

   The angle brackets are used for the symbolic grouping of a phrase representing a single parameter and conforming applications shall not include them in data submitted to the utility.

5. When a utility has only a few permissible options, they are sometimes shown individually, as in the example. Utilities with many flags generally show all of the individual flags (that do not take option-arguments) grouped, as in:

   `utility_name [-abcDxyz] [-p arg] [operand]`

   Utilities with very complex arguments may be shown as follows:

   `utility_name [options] [operands]`

6. Unless otherwise specified, whenever an operand or option-argument is, or contains, a numeric value:

   - The number is interpreted as a decimal integer.
   - Numerals in the range 0 to 2 147 483 647 are syntactically recognized as numeric values.
   - When the utility description states that it accepts negative numbers as operands or option-arguments, numerals in the range –2 147 483 647 to 2 147 483 647 are syntactically recognized as numeric values.
   - When the utility description states that the number is a file size-related value (such as a file size or offset, line number, or block count), numerals in the range 0 to the maximum file size supported by the implementation are syntactically recognized as numeric values (see XCU Section 1.5, on page 2469). Where negative values are permitted, any value in the range –(maximum file size) to the maximum file size is accepted.
   - Ranges greater than those listed here are allowed.

   This does not mean that all numbers within the allowable range are necessarily semantically correct. A standard utility that accepts an option-argument or operand that is to be interpreted as a number, and for which a range of values smaller than that shown above is permitted by the POSIX.1-2024, describes that smaller range along with the description of the option-argument or operand. If an error is generated, the utility's diagnostic message shall indicate that the value is out of the supported range, not that it is syntactically incorrect.

7. Arguments or option-arguments enclosed in the `'['` and `']'` notation are optional and can be omitted. Conforming applications shall not include the `'['` and `']'` symbols in data submitted to the utility.

8. Arguments separated by the '|' (<vertical-line>) bar notation are mutually-exclusive. Conforming applications shall not include the '|' symbol in data submitted to the utility. Alternatively, mutually-exclusive options and operands may be listed with multiple synopsis lines.

   For example:

   ```
   utility_name –d [–a] [–c option_argument] [operand...]
   utility_name [–a] [–b] [operand...]
   ```

   When multiple synopsis lines are given for a utility, it is an indication that the utility has mutually-exclusive arguments. These mutually-exclusive arguments alter the functionality of the utility so that only certain other arguments are valid in combination with one of the mutually-exclusive arguments. Only one of the mutually-exclusive arguments is allowed for invocation of the utility. Unless otherwise stated in an accompanying OPTIONS section, the relationships between arguments depicted in the SYNOPSIS sections are mandatory requirements placed on conforming applications. The use of conflicting mutually-exclusive arguments produces undefined results, unless a utility description specifies otherwise. When an option is shown without the '[' and ']' brackets, it means that option is required for that version of the SYNOPSIS. However, it is not required to be the first argument, as shown in the example above, unless otherwise stated.

9. Ellipses ("...") are used to denote that one or more occurrences of an operand are allowed. When an option or an operand followed by ellipses is enclosed in brackets, zero or more options or operands can be specified. The form:

   ```
   utility_name [–g option_argument]... [operand...]
   ```

   indicates that multiple occurrences of the option and its option-argument preceding the ellipses are valid, with semantics as indicated in the OPTIONS section of the utility. (See also Guideline 11 in Section 12.2.)

   The form:

   ```
   utility_name –f option_argument [–f option_argument]... [operand...]
   ```

   indicates that the **–f** option is required to appear at least once and may appear multiple times.

10. When the synopsis line is too long to be printed on a single line in the Shell and Utilities volume of POSIX.1-2024, the indented lines following the initial line are continuation lines. An actual use of the command would appear on a single logical line.

12.2 Utility Syntax Guidelines

The following guidelines are established for the naming of utilities and for the specification of options, option-arguments, and operands. The *getopt*() function in the System Interfaces volume of POSIX.1-2024 assists utilities in handling options and operands that conform to these guidelines.

Operands and option-arguments can contain characters not specified in the portable character set.

The guidelines are intended to provide guidance to the authors of future utilities, such as those written specific to a local system or that are components of a larger application. Some of the standard utilities do not conform to all of these guidelines; in those cases, the OPTIONS sections describe the deviations.

Guideline 1: Utility names should be between two and nine characters, inclusive.

Guideline 2: Utility names should include lowercase letters (the **lower** character classification) and digits only from the portable character set.

Guideline 3: Each option name should be a single alphanumeric character (the **alnum** character classification) from the portable character set. The –**W** (capital-W) option shall be reserved for vendor options.

Multi-digit options should not be allowed.

Guideline 4: All options should be preceded by the `'-'` delimiter character.

Guideline 5: One or more options without option-arguments, followed by at most one option that takes an option-argument, should be accepted when grouped behind one `'-'` delimiter.

Guideline 6: Each option and option-argument should be a separate argument, except as noted in Section 12.1 (on page 213), item (2).

Guideline 7: Option-arguments should not be optional.

Guideline 8: When multiple option-arguments are specified to follow a single option, they should be presented as a single argument, using <comma> characters within that argument or <blank> characters within that argument to separate them.

Guideline 9: All options should precede operands on the command line.

Guideline 10: The first –– argument that is not an option-argument should be accepted as a delimiter indicating the end of options. Any following arguments should be treated as operands, even if they begin with the `'-'` character.

Guideline 11: The order of different options relative to one another should not matter, unless the options are documented as mutually-exclusive and such an option is documented to override any incompatible options preceding it. If an option that has option-arguments is repeated, the option and option-argument combinations should be interpreted in the order specified on the command line.

Guideline 12: The order of operands may matter and position-related interpretations should be determined on a utility-specific basis.

Guideline 13: For utilities that use operands to represent files to be opened for either reading or writing, the `'-'` operand should be used to mean only standard input (or standard output when it is clear from context that an output file is being specified) or a file named –.

Guideline 14: If an argument can be identified according to Guidelines 3 through 10 as an option, or as a group of options without option-arguments behind one `'-'` delimiter, then it should be treated as such.

The utilities in the Shell and Utilities volume of POSIX.1-2024 that claim conformance to these guidelines shall conform completely to these guidelines as if these guidelines contained the term ``shall'' instead of ``should''. On some implementations, the utilities accept usage in violation of these guidelines for backwards-compatibility as well as accepting the required form.

Where a utility described in the Shell and Utilities volume of POSIX.1-2024 as conforming to these guidelines is required to accept, or not to accept, the operand `'-'` to mean standard input or output, this usage is explained in the OPERANDS section. Otherwise, if such a utility uses operands to represent files, it is implementation-defined whether the operand `'-'` stands for standard input (or standard output), or for a file named –.

It is recommended that all future utilities and applications use these guidelines to enhance user portability. The fact that some historical utilities could not be changed (to avoid breaking existing applications) should not deter this future goal.

7812 *Chapter 13*

7813 # *Namespace and Future Directions*

7814 In order to prevent future versions of this standard from introducing features that could cause
7815 older applications to fail, the prefixes posix_, POSIX_, and _POSIX_ are reserved for use by this
7816 standard for any name (e.g. function, variable, utility, etc.). Neither implementations nor
7817 applications shall introduce objects in this namespace.

Chapter 14
Headers

This chapter describes the contents of headers.

Headers contain function prototypes, the definition of symbolic constants, common structures, preprocessor macros, and defined types. Each function in the System Interfaces volume of POSIX.1-2024 specifies the headers that an application shall include in order to use that function. In most cases, only one header is required. These headers are present on an application development system; they need not be present on the target execution system.

Format of Entries

The entries in this chapter are based on a common format as follows. The only sections relating to conformance are the SYNOPSIS and DESCRIPTION.

NAME

This section gives the name or names of the entry and briefly states its purpose.

SYNOPSIS

This section summarizes the use of the entry being described.

DESCRIPTION

This section describes the functionality of the header.

APPLICATION USAGE

This section is informative. This section gives warnings and advice to application developers about the entry. In the event of conflict between warnings and advice and a normative part of this volume of POSIX.1-2024, the normative material is to be taken as correct.

RATIONALE

This section is informative. This section contains historical information concerning the contents of this volume of POSIX.1-2024 and why features were included or discarded by the standard developers.

FUTURE DIRECTIONS

This section is informative. This section provides comments which should be used as a guide to current thinking; there is not necessarily a commitment to adopt these future directions.

SEE ALSO

This section is informative. This section gives references to related information.

CHANGE HISTORY

This section is informative. This section shows the derivation of the entry and any significant changes that have been made to it.

NAME

aio.h — asynchronous input and output

SYNOPSIS

```
#include <aio.h>
```

DESCRIPTION

The **<aio.h>** header shall define the **aiocb** structure, which shall include at least the following members:

`int`	`aio_fildes`	File descriptor.
`off_t`	`aio_offset`	File offset.
`volatile void *`	`aio_buf`	Location of buffer.
`size_t`	`aio_nbytes`	Length of transfer.
`int`	`aio_reqprio`	Request priority offset.
`struct sigevent`	`aio_sigevent`	Signal number and value.
`int`	`aio_lio_opcode`	Operation to be performed.

The **<aio.h>** header shall define the **off_t**, **pthread_attr_t**, **size_t**, and **ssize_t** types as described in **<sys/types.h>**.

The **<aio.h>** header shall define the **struct timespec** structure as described in **<time.h>**.

The **<aio.h>** header shall define the **sigevent** structure and **sigval** union as described in **<signal.h>**.

The **<aio.h>** header shall define the following symbolic constants:

AIO_ALLDONE
: A return value indicating that none of the requested operations could be canceled since they are already complete.

AIO_CANCELED
: A return value indicating that all requested operations have been canceled.

AIO_NOTCANCELED
: A return value indicating that some of the requested operations could not be canceled since they are in progress.

LIO_NOP
: A *lio_listio*() element operation option indicating that no transfer is requested.

LIO_NOWAIT
: A *lio_listio*() synchronization operation indicating that the calling thread is to continue execution while the *lio_listio*() operation is being performed, and no notification is given when the operation is complete.

LIO_READ
: A *lio_listio*() element operation option requesting a read.

LIO_WAIT
: A *lio_listio*() synchronization operation indicating that the calling thread is to suspend until the *lio_listio*() operation is complete.

LIO_WRITE
: A *lio_listio*() element operation option requesting a write.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int      aio_cancel(int, struct aiocb *);
int      aio_error(const struct aiocb *);
FSC|SIO int      aio_fsync(int, struct aiocb *);
int      aio_read(struct aiocb *);
ssize_t  aio_return(struct aiocb *);
int      aio_suspend(const struct aiocb *const [], int,
```



```
                    const struct timespec *);
int         aio_write(struct aiocb *);
int         lio_listio(int, struct aiocb *restrict const [restrict], int,
                    struct sigevent *restrict);
```

Inclusion of the **<aio.h>** header may make visible symbols defined in the headers **<fcntl.h>**, **<signal.h>**, and **<time.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<fcntl.h>, **<signal.h>**, **<sys/types.h>**, **<time.h>**

XSH *aio_cancel*( ), *aio_error*( ), *aio_fsync*( ), *aio_read*( ), *aio_return*( ), *aio_suspend*( ), *aio_write*( ), *fsync*( ), *lio_listio*( ), *lseek*( ), *read*( ), *write*( )

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The **<aio.h>** header is marked as part of the Asynchronous Input and Output option.

The description of the constants is expanded.

The **restrict** keyword is added to the prototype for *lio_listio*( ).

Issue 7

The **<aio.h>** header is moved from the Asynchronous Input and Output option to the Base.

This reference page is clarified with respect to macros and symbolic constants, and type and structure declarations are added.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0038 [98] is applied.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0058 [579] is applied.

NAME

arpa/inet.h — definitions for internet operations

SYNOPSIS

```
#include <arpa/inet.h>
```

DESCRIPTION

The **<arpa/inet.h>** header shall define the **in_port_t** and **in_addr_t** types as described in **<netinet/in.h>** and the **socklen_t** type as defined in **<sys/socket.h>**.

The **<arpa/inet.h>** header shall define the **in_addr** structure as described in **<netinet/in.h>**.

IP6 The **<arpa/inet.h>** header shall define the INET_ADDRSTRLEN and INET6_ADDRSTRLEN macros as described in **<netinet/in.h>**.

The following shall be declared as functions, or defined as macros, or both. If functions are declared, function prototypes shall be provided.

```
uint32_t htonl(uint32_t);
uint16_t htons(uint16_t);
uint32_t ntohl(uint32_t);
uint16_t ntohs(uint16_t);
```

The **<arpa/inet.h>** header shall define the **uint32_t** and **uint16_t** types as described in **<inttypes.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

OB
```
in_addr_t    inet_addr(const char *);
char        *inet_ntoa(struct in_addr);
const char  *inet_ntop(int, const void *restrict, char *restrict,
                 socklen_t);
int          inet_pton(int, const char *restrict, void *restrict);
```

Inclusion of the **<arpa/inet.h>** header may also make visible all symbols from **<netinet/in.h>** and **<inttypes.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<endian.h>, **<inttypes.h>**, **<netinet/in.h>**

XSH *htonl*( ), *inet_addr*( ), *inet_ntop*( )

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The **restrict** keyword is added to the prototypes for *inet_ntop*( ) and *inet_pton*( ).

Issue 7

SD5-XBD-ERN-6 is applied.

Issue 8

Austin Group Defect 162 is applied, adding **<endian.h>** to the SEE ALSO section.

Austin Group Defects 1101 and 1102 are applied, marking *inet_addr*() and *inet_ntoa*() as obsolescent.

Austin Group Defect 1290 is applied, adding a requirement for **<arpa/inet.h>** to define the **socklen_t** type.

NAME

assert.h — verify program assertion

SYNOPSIS

```
#include <assert.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The **<assert.h>** header shall define the *assert*() macro. It refers to the macro NDEBUG which is not defined in the header. If NDEBUG is defined as a macro name before the inclusion of this header, the *assert*() macro shall be defined simply as:

```
#define assert(ignore)((void) 0)
```

Otherwise, the macro behaves as described in *assert*().

The *assert*() macro shall be redefined according to the current state of NDEBUG each time **<assert.h>** is included.

The *assert*() macro shall be implemented as a macro, not as a function. If the macro definition is suppressed in order to access an actual function, the behavior is undefined.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH *assert*()

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The definition of the *assert*() macro is changed for alignment with the ISO/IEC 9899:1999 standard.

NAME

complex.h — complex arithmetic

SYNOPSIS

```
#include <complex.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

Implementations shall not define the macro __STDC_NO_COMPLEX__, except for profile implementations that define _POSIX_SUBPROFILE (see Section 2.1.5.1, on page 20) in **<unistd.h>**, which may define __STDC_NO_COMPLEX__ and, if they do so, need not provide this header nor support any of its facilities.

The **<complex.h>** header shall define the following macros:

complex	Expands to **_Complex**.
_Complex_I	Expands to a constant expression of type **const float _Complex**, with the value of the imaginary unit (that is, a number i such that $i^2=-1$).
imaginary	Expands to **_Imaginary**.
_Imaginary_I	Expands to a constant expression of type **const float _Imaginary** with the value of the imaginary unit.
I	Expands to either _Imaginary_I or _Complex_I. If _Imaginary_I is not defined, I expands to _Complex_I.

The macros imaginary and _Imaginary_I shall be defined if and only if the implementation
MXC supports imaginary types. Implementations that support the IEC 60559 Complex Floating-Point option shall define the macros imaginary and _Imaginary_I, and the macro I shall expand to _Imaginary_I.

An application may undefine and then, perhaps, redefine the complex, imaginary, and I macros.

The following shall be defined as macros.

```
double complex       CMPLX(double x, double y);
float complex        CMPLXF(float x, float y);
long double complex  CMPLXL(long double x, long double y);
```

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
double               cabs(double complex);
float                cabsf(float complex);
long double          cabsl(long double complex);
double complex       cacos(double complex);
float complex        cacosf(float complex);
double complex       cacosh(double complex);
float complex        cacoshf(float complex);
long double complex  cacoshl(long double complex);
long double complex  cacosl(long double complex);
double               carg(double complex);
float                cargf(float complex);
long double          cargl(long double complex);
double complex       casin(double complex);
```



```
8047        float complex        casinf(float complex);
8048        double complex       casinh(double complex);
8049        float complex        casinhf(float complex);
8050        long double complex  casinhl(long double complex);
8051        long double complex  casinl(long double complex);
8052        double complex       catan(double complex);
8053        float complex        catanf(float complex);
8054        double complex       catanh(double complex);
8055        float complex        catanhf(float complex);
8056        long double complex  catanhl(long double complex);
8057        long double complex  catanl(long double complex);
8058        double complex       ccos(double complex);
8059        float complex        ccosf(float complex);
8060        double complex       ccosh(double complex);
8061        float complex        ccoshf(float complex);
8062        long double complex  ccoshl(long double complex);
8063        long double complex  ccosl(long double complex);
8064        double complex       cexp(double complex);
8065        float complex        cexpf(float complex);
8066        long double complex  cexpl(long double complex);
8067        double               cimag(double complex);
8068        float                cimagf(float complex);
8069        long double          cimagl(long double complex);
8070        double complex       clog(double complex);
8071        float complex        clogf(float complex);
8072        long double complex  clogl(long double complex);
8073        double complex       conj(double complex);
8074        float complex        conjf(float complex);
8075        long double complex  conjl(long double complex);
8076        double complex       cpow(double complex, double complex);
8077        float complex        cpowf(float complex, float complex);
8078        long double complex  cpowl(long double complex, long double complex);
8079        double complex       cproj(double complex);
8080        float complex        cprojf(float complex);
8081        long double complex  cprojl(long double complex);
8082        double               creal(double complex);
8083        float                crealf(float complex);
8084        long double          creall(long double complex);
8085        double complex       csin(double complex);
8086        float complex        csinf(float complex);
8087        double complex       csinh(double complex);
8088        float complex        csinhf(float complex);
8089        long double complex  csinhl(long double complex);
8090        long double complex  csinl(long double complex);
8091        double complex       csqrt(double complex);
8092        float complex        csqrtf(float complex);
8093        long double complex  csqrtl(long double complex);
8094        double complex       ctan(double complex);
8095        float complex        ctanf(float complex);
8096        double complex       ctanh(double complex);
8097        float complex        ctanhf(float complex);
8098        long double complex  ctanhl(long double complex);
```



```
long double complex  ctanl(long double complex);
```

APPLICATION USAGE

The **<complex.h>** header is optional in the ISO C standard but is mandated by POSIX.1-2024. Note however that subprofiles can choose to make this header optional (see Section 2.1.5.1, on page 20), and therefore application portability to subprofile implementations would benefit from checking whether __STDC_NO_COMPLEX__ is defined before inclusion of **<complex.h>**.

Values are interpreted as radians, not degrees.

RATIONALE

The choice of *I* instead of *i* for the imaginary unit concedes to the widespread use of the identifier *i* for other purposes. The application can use a different identifier, say *j*, for the imaginary unit by following the inclusion of the **<complex.h>** header with:

```
#undef I
#define j _Imaginary_I
```

An *I* suffix to designate imaginary constants is not required, as multiplication by *I* provides a sufficiently convenient and more generally useful notation for imaginary terms. The corresponding real type for the imaginary unit is **float**, so that use of *I* for algorithmic or notational convenience will not result in widening types.

On systems with imaginary types, the application has the ability to control whether use of the macro I introduces an imaginary type, by explicitly defining I to be _Imaginary_I or _Complex_I. Disallowing imaginary types is useful for some applications intended to run on implementations without support for such types.

The macro _Imaginary_I provides a test for whether imaginary types are supported.

The *cis*( ) function (*cos*(*x*) + *I***sin*(*x*)) was considered but rejected because its implementation is easy and straightforward, even though some implementations could compute sine and cosine more efficiently in tandem.

FUTURE DIRECTIONS

The following function names and the same names suffixed with *f* or *l* are reserved for future use, and may be added to the declarations in the **<complex.h>** header.

cerf( )	*cexpm1*( )	*clog2*( )
cerfc( )	*clog10*( )	*clgamma*( )
cexp2( )	*clog1p*( )	*ctgamma*( )

SEE ALSO

XSH *CMPLX*( ), *cabs*( ), *cacos*( ), *cacosh*( ), *carg*( ), *casin*( ), *casinh*( ), *catan*( ), *catanh*( ), *ccos*( ), *ccosh*( ), *cexp*( ), *cimag*( ), *clog*( ), *conj*( ), *cpow*( ), *cproj*( ), *creal*( ), *csin*( ), *csinh*( ), *csqrt*( ), *ctan*( ), *ctanh*( )

CHANGE HISTORY

First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899: 2018 standard.

NAME

cpio.h — cpio archive values

SYNOPSIS

```
#include <cpio.h>
```

DESCRIPTION

The **<cpio.h>** header shall define the symbolic constants needed by the *c_mode* field of the *cpio* archive format, with the names and values given in the following table:

Name	Description	Value (Octal)
C_IRUSR	Read by owner.	0000400
C_IWUSR	Write by owner.	0000200
C_IXUSR	Execute by owner.	0000100
C_IRGRP	Read by group.	0000040
C_IWGRP	Write by group.	0000020
C_IXGRP	Execute by group.	0000010
C_IROTH	Read by others.	0000004
C_IWOTH	Write by others.	0000002
C_IXOTH	Execute by others.	0000001
C_ISUID	Set user ID.	0004000
C_ISGID	Set group ID.	0002000
C_ISVTX	On directories, restricted deletion flag.	0001000
C_ISDIR	Directory.	0040000
C_ISFIFO	FIFO.	0010000
C_ISREG	Regular file.	0100000
C_ISBLK	Block special.	0060000
C_ISCHR	Character special.	0020000
C_ISCTG	Reserved.	0110000
C_ISLNK	Symbolic link.	0120000
C_ISSOCK	Socket.	0140000

The **<cpio.h>** header shall define the following symbolic constant as a string:

```
MAGIC    "070707"
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XCU *pax*

CHANGE HISTORY

First released in the Headers Interface, Issue 3 specification. Derived from the POSIX.1-1988 standard.

Issue 6

The SEE ALSO is updated to refer to *pax*.

8180 **Issue 7**

8181 The **<cpio.h>** header is moved from the XSI option to the Base.

8182 This reference page is clarified with respect to macros and symbolic constants.

NAME

ctype.h — character types

SYNOPSIS

```
#include <ctype.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to enable the visibility of these symbols in this header.

The **<ctype.h>** header shall define the **locale_t** type as described in **<locale.h>**, representing a locale object.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

```
int   isalnum(int);
CX int   isalnum_l(int, locale_t);
int   isalpha(int);
CX int   isalpha_l(int, locale_t);
int   isblank(int);
CX int   isblank_l(int, locale_t);
int   iscntrl(int);
CX int   iscntrl_l(int, locale_t);
int   isdigit(int);
CX int   isdigit_l(int, locale_t);
int   isgraph(int);
CX int   isgraph_l(int, locale_t);
int   islower(int);
CX int   islower_l(int, locale_t);
int   isprint(int);
CX int   isprint_l(int, locale_t);
int   ispunct(int);
CX int   ispunct_l(int, locale_t);
int   isspace(int);
CX int   isspace_l(int, locale_t);
int   isupper(int);
CX int   isupper_l(int, locale_t);
int   isxdigit(int);
CX int   isxdigit_l(int, locale_t);
int   tolower(int);
CX int   tolower_l(int, locale_t);
int   toupper(int);
CX int   toupper_l(int, locale_t);
```


8223 **APPLICATION USAGE**

8224 None.

8225 **RATIONALE**

8226 None.

8227 **FUTURE DIRECTIONS**

8228 None.

8229 **SEE ALSO**

8230 **<locale.h>**

8231 XSH Section 2.2 (on page 496), *isalnum*( ), *isalpha*( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ),
8232 *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *mblen*( ), *mbstowcs*( ), *mbtowc*( ),
8233 *setlocale*( ), *tolower*( ), *toupper*( ), *wcstombs*( ), *wctomb*( )

8234 **CHANGE HISTORY**

8235 First released in Issue 1. Derived from Issue 1 of the SVID.

8236 **Issue 6**

8237 Extensions beyond the ISO C standard are marked.

8238 **Issue 7**

8239 SD5-XBD-ERN-6 is applied, updating the wording regarding the function declarations for
8240 consistency.

8241 The **_l*( ) functions are added from The Open Group Technical Standard, 2006, Extended API Set
8242 Part 4.

8243 **Issue 8**

8244 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

NAME

devctl.h — device control

SYNOPSIS

DC

```
#include <devctl.h>
```

DESCRIPTION

The **<devctl.h>** header shall define the **size_t** type as described in **<sys/types.h>**.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
int posix_devctl(int, int, void *restrict, size_t, int *restrict);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, **<termios.h>**

XSH *posix_devctl( )*

CHANGE HISTORY

First released in Issue 8. Derived from POSIX.26.

NAME

dirent.h — format of directory entries

SYNOPSIS

```
#include <dirent.h>
```

DESCRIPTION

The internal format of directories is unspecified.

The **<dirent.h>** header shall define the following type:

DIR A type representing a directory stream. The **DIR** type may be an incomplete type.

It shall also define the structure **dirent** which shall include the following members:

`ino_t`	`d_ino`	File serial number.
`char`	`d_name[]`	Filename string of entry.

and the structure **posix_dent** which shall include the following members:

`ino_t`	`d_ino`	File serial number.
`reclen_t`	`d_reclen`	Length of this entry, including trailing padding if necessary. See *posix_getdents*( ).
`unsigned char`	`d_type`	File type or unknown-file-type indication.
`char`	`d_name[]`	Filename string of this entry.

The array *d_name* in each of these structures is of unspecified size, but shall contain a filename of at most {NAME_MAX} bytes followed by a terminating null byte.

The **<dirent.h>** header shall define the **ino_t**, **reclen_t**, **size_t**, and **ssize_t** types as described in **<sys/types.h>**.

The **<dirent.h>** header shall define the following symbolic constants for the file types and unknown-file-type indicator returned in the *d_type* member of the **posix_dent** structure. The values shall be distinct and shall be suitable for use in **#if** preprocessing directives:

DT_BLK Block special.

DT_CHR Character special.

DT_DIR Directory.

DT_FIFO FIFO special.

DT_LNK Symbolic link.

DT_REG Regular.

DT_SOCK Socket.

DT_UNKNOWN
Unknown file type.

TYM The implementation may implement message queues, semaphores, shared memory objects or typed memory objects as distinct file types. The following macros shall be provided to represent these types. The values shall be distinct from each other and from the above symbolic constants beginning with DT_, except when a distinct file type is not implemented, in which case the corresponding constant shall have a value that is never returned in *d_type* by *posix_getdents*( ). The values shall be suitable for use in **#if** preprocessing directives:

DT_MQ Message queue.

DT_SEM Semaphore.

DT_SHM Shared memory object.

TYM DT_TMO Typed memory object.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int             alphasort(const struct dirent **, const struct dirent **);
int             closedir(DIR *);
int             dirfd(DIR *);
DIR            *fdopendir(int);
DIR            *opendir(const char *);
ssize_t         posix_getdents(int, void *, size_t, int);
struct dirent  *readdir(DIR *);
OB int          readdir_r(DIR *restrict, struct dirent *restrict,
                    struct dirent **restrict);
void            rewinddir(DIR *);
int             scandir(const char *, struct dirent ***,
                    int (*)(const struct dirent *),
                    int (*)(const struct dirent **,
                    const struct dirent **));
XSI void        seekdir(DIR *, long);
long            telldir(DIR *);
```

APPLICATION USAGE

None.

RATIONALE

Information similar to that in the **<dirent.h>** header is contained in a file **<sys/dir.h>** in 4.2 BSD and 4.3 BSD. The equivalent in these implementations of **struct dirent** from this volume of POSIX.1-2024 is **struct direct**. The filename was changed because the name **<sys/dir.h>** was also used in earlier implementations to refer to definitions related to the older access method; this produced name conflicts. The name of the structure was changed because this volume of POSIX.1-2024 does not completely define what is in the structure, so it could be different on some implementations from **struct direct**.

The **posix_dent** structure was based on existing structures used by traditional *getdents*() functions, but the name was changed because the existing structures differed in name and in their members. Some used the **dirent** structure but this is not required to include a *d_type* member, which is the main advantage of using *posix_getdents*() over *readdir*(). The *d_reclen* member was included, even though some implementations return fixed-length entries and therefore do not need it, as almost all existing code that used *getdents*() used *d_reclen* to iterate through the returned entries. Implementations that return fixed-length entries can simply set *d_reclen* to that length in *posix_getdents*(). The type **reclen_t** for *d_reclen* was introduced, instead of using **unsigned short**, so as not to create a requirement that {NAME_MAX} cannot be greater than (a value somewhat smaller than) {SHRT_MAX}.

Implementations are encouraged to define a DT_FORCE_TYPE symbolic constant for use in the *flags* argument to *posix_getdents*(). See the RATIONALE for *posix_getdents*().

The name of an array of **char** of an unspecified size should not be used as an lvalue. Use of:

```
sizeof(d_name)
```


is incorrect; use:

```
strlen(d_name)
```

instead.

The array of **char** *d_name* cannot be assumed to have a fixed size. Implementations may define the *d_name* array in the **dirent** and **posix_dent** structures to have size 1, or size greater than {NAME_MAX}, or use a flexible array member, but in all cases the actual number of characters used for *d_name* is at least the length of the filename string including the terminating NUL byte.

FUTURE DIRECTIONS

A future version of this standard may add a DT_FORCE_TYPE symbolic constant for use as described in the RATIONALE for *posix_getdents*( ).

SEE ALSO

<sys/types.h>

XSH *alphasort*( ), *closedir*( ), *dirfd*( ), *fdopendir*( ), *posix_getdents*( ), *readdir*( ), *rewinddir*( ), *seekdir*( ), *telldir*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir_r*( ).

The **restrict** keyword is added to the prototype for *readdir_r*( ).

Issue 7

The *alphasort*( ), *dirfd*( ), and *scandir*( ) functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The *fdopendir*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Austin Group Interpretation 1003.1-2001 #110 is applied, clarifying the definition of the **DIR** type.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0039 [291], XBD/TC1-2008/0040 [291], XBD/TC1-2008/0041 [291], and XBD/TC1-2008/0042 [206] are applied.

Issue 8

Austin Group Defect 696 is applied, making *readdir_r*( ) obsolescent.

Austin Group Defect 697 is applied, adding *posix_getdents*( ).

NAME

dlfcn.h — dynamic linking

SYNOPSIS

```
#include <dlfcn.h>
```

DESCRIPTION

The **<dlfcn.h>** header shall define the **Dl_info_t** structure type, which shall include at least the following members:

```
const char *dli_fname     Pathname of mapped object file.
void       *dli_fbase     Base of mapped address range.
const char *dli_sname     Symbol name or null pointer.
void       *dli_saddr     Symbol address or null pointer.
```

The **<dlfcn.h>** header shall define at least the following symbolic constants for use in the construction of a *dlopen*( ) *mode* argument:

RTLD_LAZY — Relocations are performed at an implementation-defined time.

RTLD_NOW — Relocations are performed when the object is loaded.

RTLD_GLOBAL — All symbols are available for relocation processing of other modules.

RTLD_LOCAL — All symbols are not made available for relocation processing by other modules.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int    dladdr(const void *restrict, Dl_info_t *restrict);
int    dlclose(void *);
char  *dlerror(void);
void  *dlopen(const char *, int);
void  *dlsym(void *restrict, const char *restrict);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH *dladdr*( ), *dlclose*( ), *dlerror*( ), *dlopen*( ), *dlsym*( )

CHANGE HISTORY

First released in Issue 5.

Issue 6

The **restrict** keyword is added to the prototype for *dlsym*( ).

Issue 7

The **<dlfcn.h>** header is moved from the XSI option to the Base.

This reference page is clarified with respect to macros and symbolic constants.

8423 **Issue 8**

8424 Austin Group Defect 993 is applied, adding *dladdr*( ).

NAME

endian.h — system endianness

SYNOPSIS

```
#include <endian.h>
```

DESCRIPTION

The **<endian.h>** header shall define at least the following macros for use in determining host byte order for integer types.

BYTE_ORDER This macro shall have a value equal to one of the *_ENDIAN macros in this header.

LITTLE_ENDIAN
If BYTE_ORDER == LITTLE_ENDIAN, the host byte order is from least significant to most significant.

BIG_ENDIAN If BYTE_ORDER == BIG_ENDIAN, the host byte order is from most significant to least significant.

These macros shall be suitable for use in **#if** preprocessing directives. The macros BIG_ENDIAN and LITTLE_ENDIAN shall have distinct values. Implementations may define other macros with the _ENDIAN suffix.

The following shall be declared as functions, or defined as macros, or both. If functions are declared, function prototypes shall be provided.

```
uint16_t  be16toh(uint16_t);
uint32_t  be32toh(uint32_t);
uint64_t  be64toh(uint64_t);

uint16_t  htobe16(uint16_t);
uint32_t  htobe32(uint32_t);
uint64_t  htobe64(uint64_t);

uint16_t  htole16(uint16_t);
uint32_t  htole32(uint32_t);
uint64_t  htole64(uint64_t);

uint16_t  le16toh(uint16_t);
uint32_t  le32toh(uint32_t);
uint64_t  le64toh(uint64_t);
```

The **<endian.h>** header shall define the **uint16_t**, **uint32_t**, and **uint64_t** types as described in **<stdint.h>**.

Inclusion of the **<endian.h>** header may also make visible all symbols from **<stdint.h>**.

APPLICATION USAGE

None.

RATIONALE

Many implementations also include PDP_ENDIAN to indicate a byte ordering where each pair of bytes is swapped. If BIG_ENDIAN is defined as 4321, PDP_ENDIAN would be 3412. However, this scheme is not universal, and derives its name from an obsolete processor.

FUTURE DIRECTIONS

None.

8467 **SEE ALSO**

8468 **<stdint.h>**

8469 XSH *be16toh*( ), *htonl*( ), *swab*( )

8470 **CHANGE HISTORY**

8471 First released in Issue 8.

NAME

errno.h — system error numbers

SYNOPSIS

```
#include <errno.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The ISO C standard only requires the symbols [EDOM], [EILSEQ], and [ERANGE] to be defined.

The **<errno.h>** header shall provide a definition for the macro *errno*, which shall expand to a modifiable lvalue of type **int** and thread local storage duration. If the macro definition is suppressed in order to access an actual object, or a program defines an identifier with the name *errno*, the behavior is undefined.

The **<errno.h>** header shall define the following macros which shall expand to integer constant expressions with type **int**, distinct positive values (except as noted below), and which shall be suitable for use in **#if** preprocessing directives:

[E2BIG]	Argument list too long.
[EACCES]	Permission denied.
[EADDRINUSE]	Address in use.
[EADDRNOTAVAIL]	Address not available.
[EAFNOSUPPORT]	Address family not supported.
[EAGAIN]	Resource unavailable, try again (may be the same value as [EWOULDBLOCK]).
[EALREADY]	Connection already in progress.
[EBADF]	Bad file descriptor.
[EBADMSG]	Bad message.
[EBUSY]	Device or resource busy.
[ECANCELED]	Operation canceled.
[ECHILD]	No child processes.
[ECONNABORTED]	Connection aborted.
[ECONNREFUSED]	Connection refused.
[ECONNRESET]	Connection reset.
[EDEADLK]	Resource deadlock would occur.
[EDESTADDRREQ]	Destination address required.
[EDOM]	Mathematics argument out of domain of function.
[EDQUOT]	Reserved.
[EEXIST]	File exists.

[EFAULT]	Bad address.
[EFBIG]	File too large.
[EHOSTUNREACH]	Host is unreachable.
[EIDRM]	Identifier removed.
[EILSEQ]	Illegal byte sequence.
[EINPROGRESS]	Operation in progress.
[EINTR]	Interrupted function.
[EINVAL]	Invalid argument.
[EIO]	I/O error.
[EISCONN]	Socket is connected.
[EISDIR]	Is a directory.
[ELOOP]	Too many levels of symbolic links.
[EMFILE]	File descriptor value too large.
[EMLINK]	Too many hard links.
[EMSGSIZE]	Message too large.
[EMULTIHOP]	Reserved.
[ENAMETOOLONG]	Filename too long.
[ENETDOWN]	Network is down.
[ENETRESET]	Connection aborted by network.
[ENETUNREACH]	Network unreachable.
[ENFILE]	Too many files open in system.
[ENOBUFS]	No buffer space available.
[ENODEV]	No such device.
[ENOENT]	No such file or directory.
[ENOEXEC]	Executable file format error.
[ENOLCK]	No locks available.
[ENOLINK]	Reserved.
[ENOMEM]	Not enough space.
[ENOMSG]	No message of the desired type.
[ENOPROTOOPT]	Protocol not available.
[ENOSPC]	No space left on device.
[ENOSYS]	Functionality not supported.
[ENOTCONN]	The socket is not connected.

8542 [ENOTDIR] Not a directory or a symbolic link to a directory.

8543 [ENOTEMPTY] Directory not empty.

8544 [ENOTRECOVERABLE]
8545 State not recoverable.

8546 [ENOTSOCK] Not a socket.

8547 [ENOTSUP] Not supported (may be the same value as [EOPNOTSUPP]).

8548 [ENOTTY] Inappropriate I/O control operation.

8549 [ENXIO] No such device or address.

8550 [EOPNOTSUPP] Operation not supported on socket (may be the same value as
8551 [ENOTSUP]).

8552 [EOVERFLOW] Value too large to be stored in data type.

8553 [EOWNERDEAD] Previous owner died.

8554 [EPERM] Operation not permitted.

8555 [EPIPE] Broken pipe.

8556 [EPROTO] Protocol error.

8557 [EPROTONOSUPPORT]
8558 Protocol not supported.

8559 [EPROTOTYPE] Protocol wrong type for socket.

8560 [ERANGE] Result too large.

8561 [EROFS] Read-only file system.

8562 [ESOCKTNOSUPPORT]
8563 Socket type not supported.

8564 [ESPIPE] Invalid seek.

8565 [ESRCH] No such process.

8566 [ESTALE] Reserved.

8567 [ETIMEDOUT] Connection timed out.

8568 [ETXTBSY] Text file busy.

8569 [EWOULDBLOCK] Operation would block (may be the same value as [EAGAIN]).

8570 [EXDEV] Improper hard link.

8571 **APPLICATION USAGE**

8572 Additional error numbers may be defined on conforming systems; see the System Interfaces
8573 volume of POSIX.1-2024.

8574 **RATIONALE**

8575 None.

8576 **FUTURE DIRECTIONS**

8577 None.

8578 **SEE ALSO**

8579 XSH Section 2.3 (on page 507)

8580 **CHANGE HISTORY**

8581 First released in Issue 1. Derived from Issue 1 of the SVID.

8582 **Issue 5**

8583 Updated for alignment with the POSIX Realtime Extension.

8584 **Issue 6**

8585 The following new requirements on POSIX implementations derive from alignment with the
8586 Single UNIX Specification:

8587 • The majority of the error conditions previously marked as extensions are now mandatory,
8588 except for the STREAMS-related error conditions.

8589 Values for *errno* are now required to be distinct positive values rather than non-zero values. This
8590 change is for alignment with the ISO/IEC 9899: 1999 standard.

8591 **Issue 7**

8592 Austin Group Interpretation 1003.1-2001 #050 is applied, allowing [ENOTSUP] and
8593 [EOPNOTSUPP] to be the same values.

8594 The [ENOTRECOVERABLE] and [EOWNERDEAD] errors are added from The Open Group
8595 Technical Standard, 2006, Extended API Set Part 2.

8596 Functionality relating to the XSI STREAMS option is marked obsolescent.

8597 Functionality relating to the Threads option is moved to the Base.

8598 This reference page is clarified with respect to macros and symbolic constants.

8599 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0043 [324] is applied.

8600 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0059 [496] is applied.

8601 **Issue 8**

8602 Austin Group Defect 1067 is applied, adding [ESOCKTNOSUPPORT].

8603 Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899: 2018 standard.

8604 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

8605 Austin Group Defect 1380 is applied, changing the descriptions of [EMLINK] and [EXDEV].

NAME

fcntl.h — file control options

SYNOPSIS

```
#include <fcntl.h>
```

DESCRIPTION

The **<fcntl.h>** header shall define the **f_owner_ex** structure, which shall include at least the following members:

`int type`	Discriminator for *pid*.
`pid_t pid`	Process ID or process group ID.

The **<fcntl.h>** header shall define the **flock** structure describing a file lock. It shall include the following members:

`short l_type`	Type of lock; F_RDLCK, F_WRLCK, F_UNLCK.
`short l_whence`	Flag for starting offset.
`off_t l_start`	Relative offset in bytes.
`off_t l_len`	Size; if 0 then until EOF.
`pid_t l_pid`	For a process-owned file lock, ignored on input or the process ID of the owning process on output; for an OFD-owned file lock, zero on input or (**pid_t**)–1 on output.

The **<fcntl.h>** header shall define the **mode_t**, **off_t**, and **pid_t** types as described in **<sys/types.h>**.

The **<fcntl.h>** header shall define the following symbolic constants for the *cmd* argument used by *fcntl*( ). The values shall be unique and shall be suitable for use in **#if** preprocessing directives.

F_DUPFD	Duplicate file descriptor.
F_DUPFD_CLOEXEC	Duplicate file descriptor with the close-on-*exec* flag FD_CLOEXEC set.
F_DUPFD_CLOFORK	Duplicate file descriptor with the close-on-*fork* flag FD_CLOFORK set.
F_GETFD	Get file descriptor flags.
F_SETFD	Set file descriptor flags.
F_GETFL	Get file status flags and file access modes.
F_SETFL	Set file status flags.
F_GETLK	Get information about file locks.
F_SETLK	Set a process-owned file lock.
F_SETLKW	Set a process-owned file lock; wait if blocked.
F_OFD_GETLK	Get information about file locks.
F_OFD_SETLK	Set an OFD-owned file lock.
F_OFD_SETLKW	Set an OFD-owned file lock; wait if blocked.
F_GETOWN	Get process or process group ID to receive SIGURG signals, via **int** type.

F_GETOWN_EX Get process or process group ID to receive SIGURG signals, via **pid_t** type.

F_SETOWN Set process or process group ID to receive SIGURG signals, via **int** type.

F_SETOWN_EX Set process or process group ID to receive SIGURG signals, via **pid_t** type.

The **<fcntl.h>** header shall define the following symbolic constants used for the *fcntl*() file descriptor flags. The values shall be bitwise-distinct and shall be suitable for use in **#if** preprocessing directives.

SPN FD_CLOEXEC Close the file descriptor upon execution of an *exec* family function and in the new process image created by *posix_spawn*() or *posix_spawnp*().

FD_CLOFORK Close the file descriptor in any child process created from a process that has the file descriptor open; that is, the child shall not inherit the file descriptor.

The **<fcntl.h>** header shall also define the following symbolic constants for the *l_type* argument used for record locking with *fcntl*(). The values shall be unique and shall be suitable for use in **#if** preprocessing directives.

F_RDLCK Shared or read lock.

F_UNLCK Unlock.

F_WRLCK Exclusive or write lock.

The **<fcntl.h>** header shall also define the following symbolic constants for the *type* member of the **f_owner_ex** structure. The values shall be unique.

F_OWNER_PID The *pid* member of **f_owner_ex** holds a process ID.

F_OWNER_PGRP
The *pid* member of **f_owner_ex** holds a process group ID.

The **<fcntl.h>** header shall define the values used for *l_whence*, SEEK_SET, SEEK_CUR, and SEEK_END as described in **<stdio.h>**.

The **<fcntl.h>** header shall define the following symbolic constants as file creation flags for use in the *oflag* value to *open*() and *openat*(). The values shall be bitwise-distinct and shall be suitable for use in **#if** preprocessing directives.

O_CLOEXEC Atomically set the FD_CLOEXEC flag on the new file descriptor.

O_CLOFORK Atomically set the FD_CLOFORK flag on the new file descriptor.

O_CREAT Create file if it does not exist.

O_DIRECTORY Fail if file is a non-directory file.

O_EXCL Exclusive use flag.

O_NOCTTY Do not assign controlling terminal.

O_NOFOLLOW Do not follow symbolic links.

O_TRUNC Truncate flag.

O_TTY_INIT Set the **termios** structure terminal parameters to a state that provides conforming behavior; see Section 11.2 (on page 205).

The O_TTY_INIT flag can have the value zero and in this case it need not be bitwise-distinct from the other flags.

The **<fcntl.h>** header shall define the following symbolic constants for use as file status flags for

open( ), *openat*( ), and *fcntl*( ). The values shall be suitable for use in **#if** preprocessing directives.

O_APPEND Set append mode.

SIO O_DSYNC Write according to synchronized I/O data integrity completion.

O_NONBLOCK Non-blocking mode.

SIO O_RSYNC Synchronized read I/O operations.

O_SYNC Write according to synchronized I/O file integrity completion.

The **<fcntl.h>** header shall define the following symbolic constant for use as the mask for file access modes. The value shall be suitable for use in **#if** preprocessing directives.

O_ACCMODE Mask for file access modes.

The **<fcntl.h>** header shall define the following symbolic constants for use as the file access modes for *open*( ), *openat*( ), and *fcntl*( ). The values shall be unique, except that O_EXEC and O_SEARCH may have equal values. The values shall be suitable for use in **#if** preprocessing directives.

O_EXEC Open for execute only (non-directory files). The result is unspecified if this flag is applied to a directory.

O_RDONLY Open for reading only.

O_RDWR Open for reading and writing.

O_SEARCH Open directory for search only. The result is unspecified if this flag is applied to a non-directory file.

O_WRONLY Open for writing only.

The **<fcntl.h>** header shall define the symbolic constants for file modes for use as values of **mode_t** as described in **<sys/stat.h>**.

The **<fcntl.h>** header shall define the following symbolic constant as a special value used in place of a file descriptor for the **at*( ) functions which take a directory file descriptor as a parameter:

AT_FDCWD Use the current working directory to determine the target of relative file paths.

The **<fcntl.h>** header shall define the following symbolic constant as a value for the *flag* used by *faccessat*( ):

AT_EACCESS Check access using effective user and group ID.

The **<fcntl.h>** header shall define the following symbolic constant as a value for the *flag* used by *fstatat*( ), *fchmodat*( ), *fchownat*( ), and *utimensat*( ):

AT_SYMLINK_NOFOLLOW
Do not follow symbolic links.

The **<fcntl.h>** header shall define the following symbolic constant as a value for the flag used by *linkat*( ):

AT_SYMLINK_FOLLOW
Follow symbolic link.

The **<fcntl.h>** header shall define the following symbolic constant as a value for the flag used by *unlinkat*( ):

AT_REMOVEDIR
: Remove directory instead of file.

ADV The **<fcntl.h>** header shall define the following symbolic constants for the *advice* argument used by *posix_fadvise*( ):

POSIX_FADV_DONTNEED
: The application expects that it will not access the specified data in the near future.

POSIX_FADV_NOREUSE
: The application expects to access the specified data once and then not reuse it thereafter.

POSIX_FADV_NORMAL
: The application has no advice to give on its behavior with respect to the specified data. It is the default characteristic if no advice is given for an open file.

POSIX_FADV_RANDOM
: The application expects to access the specified data in a random order.

POSIX_FADV_SEQUENTIAL
: The application expects to access the specified data sequentially from lower offsets to higher offsets.

POSIX_FADV_WILLNEED
: The application expects to access the specified data in the near future.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int  creat(const char *, mode_t);
int  fcntl(int, int, ...);
int  open(const char *, int, ...);
int  openat(int, const char *, int, ...);
ADV int  posix_fadvise(int, off_t, off_t, int);
int  posix_fallocate(int, off_t, off_t);
```

Inclusion of the **<fcntl.h>** header may also make visible all symbols from **<sys/stat.h>** and **<unistd.h>**.

APPLICATION USAGE

Although no existing implementation defines AT_SYMLINK_FOLLOW and AT_SYMLINK_NOFOLLOW as the same numeric value, POSIX.1-2024 does not prohibit that as the two constants are not used with the same interfaces.

RATIONALE

While many of the symbolic constants introduced in the **<fcntl.h>** header do not strictly need to be used in **#if** preprocessor directives, widespread historic practice has defined them as macros that are usable in such constructs, and examination of existing applications has shown that they are occasionally used in such a way. Therefore it was decided to retain this requirement on an implementation in POSIX.1-2024.

FUTURE DIRECTIONS

A future version of this standard may add an O_NOCLOBBER file creation flag. See the FUTURE DIRECTIONS section for *open*( ).

SEE ALSO

<stdio.h>, **<sys/stat.h>**, **<sys/types.h>**, **<unistd.h>**

XSH *creat*( ), *exec*, *fcntl*( ), *futimens*( ), *open*( ), *posix_fadvise*( ), *posix_fallocate*( ), *posix_madvise*( )

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

Issue 6

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- O_DSYNC and O_RSYNC are marked as part of the Synchronized Input and Output option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The definition of the **mode_t**, **off_t**, and **pid_t** types is mandated.

The F_GETOWN and F_SETOWN values are added for sockets.

The *posix_fadvise*( ), *posix_fallocate*( ), and *posix_madvise*( ) functions are added for alignment with IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #102 is applied, moving the prototype for *posix_madvise*( ) to **<sys/mman.h>**.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/18 is applied, updating the prototypes for *posix_fadvise*( ) and *posix_fallocate*( ) to be large file-aware, using **off_t** instead of **size_t**.

Issue 7

Austin Group Interpretation 1003.1-2001 #144 is applied, adding the O_TTY_INIT flag.

Austin Group Interpretation 1003.1-2001 #171 is applied, adding support to set the FD_CLOEXEC flag atomically at *open*( ), and adding the F_DUPFD_CLOEXEC flag.

The *openat*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Additional flags are added to support *faccessat*( ), *fchmodat*( ), *fchownat*( ), *fstatat*( ), *linkat*( ), *open*( ), *openat*( ), and *unlinkat*( ).

This reference page is clarified with respect to macros and symbolic constants.

Changes are made related to support for finegrained timestamps.

Changes are made to allow a directory to be opened for searching.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0044 [274] and XBD/TC1-2008/0045 [78,432] are applied.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0060 [847] is applied.

Issue 8

Austin Group Defect 768 is applied, adding OFD-owned file locks.

Austin Group Defect 1016 is applied, changing the FUTURE DIRECTIONS section.

Austin Group Defect 1274 is applied, adding the **f_owner_ex** structure and related symbolic constants.

Austin Group Defect 1318 is applied, adding FD_CLOFORK and O_CLOFORK, and changing O_CLOEXEC.

Austin Group Defect 1351 is applied, adding F_DUPFD_CLOFORK.

NAME

fenv.h — floating-point environment

SYNOPSIS

```
#include <fenv.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The **<fenv.h>** header shall define the following data types through **typedef**:

fenv_t Represents the entire floating-point environment. The floating-point environment refers collectively to any floating-point status flags and control modes supported by the implementation.

fexcept_t Represents the floating-point status flags collectively, including any status the implementation associates with the flags. A floating-point status flag is a system variable whose value is set (but never cleared) when a floating-point exception is raised, which occurs as a side-effect of exceptional floating-point arithmetic to provide auxiliary information. A floating-point control mode is a system variable whose value may be set by the user to affect the subsequent behavior of floating-point arithmetic.

The **<fenv.h>** header shall define each of the following macros if and only if the implementation supports the floating-point exception by means of the floating-point functions *feclearexcept*(), *fegetexceptflag*(), *feraiseexcept*(), *fesetexceptflag*(), and *fetestexcept*(). The defined macros shall expand to integer constant expressions with values that are bitwise-distinct.

FE_DIVBYZERO
FE_INEXACT
FE_INVALID
FE_OVERFLOW
FE_UNDERFLOW

MX If the implementation supports the IEC 60559 Floating-Point option, all five macros shall be defined. Additional implementation-defined floating-point exceptions with macros beginning with FE_ and an uppercase letter may also be specified by the implementation.

The **<fenv.h>** header shall define the macro FE_ALL_EXCEPT as the bitwise-inclusive OR of all floating-point exception macros defined by the implementation, if any. If no such macros are defined, then the macro FE_ALL_EXCEPT shall be defined as zero.

The **<fenv.h>** header shall define each of the following macros if and only if the implementation supports getting and setting the represented rounding direction by means of the *fegetround*() and *fesetround*() functions. The defined macros shall expand to integer constant expressions whose values are distinct non-negative values.

FE_DOWNWARD
FE_TONEAREST
FE_TOWARDZERO
FE_UPWARD

MX If the implementation supports the IEC 60559 Floating-Point option, all four macros shall be defined. Additional implementation-defined rounding directions with macros beginning with FE_ and an uppercase letter may also be specified by the implementation.

The **<fenv.h>** header shall define the following macro, which represents the default floating-point environment (that is, the one installed at program startup) and has type pointer to const-qualified **fenv_t**. It can be used as an argument to the functions within the **<fenv.h>** header that manage the floating-point environment.

FE_DFL_ENV

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int  feclearexcept(int);
int  fegetenv(fenv_t *);
int  fegetexceptflag(fexcept_t *, int);
int  fegetround(void);
int  feholdexcept(fenv_t *);
int  feraiseexcept(int);
int  fesetenv(const fenv_t *);
int  fesetexceptflag(const fexcept_t *, int);
int  fesetround(int);
int  fetestexcept(int);
int  feupdateenv(const fenv_t *);
```

The FENV_ACCESS pragma provides a means to inform the implementation when an application might access the floating-point environment to test floating-point status flags or run under non-default floating-point control modes. The pragma shall occur either outside external declarations or preceding all explicit declarations and statements inside a compound statement. When outside external declarations, the pragma takes effect from its occurrence until another FENV_ACCESS pragma is encountered, or until the end of the translation unit. When inside a compound statement, the pragma takes effect from its occurrence until another FENV_ACCESS pragma is encountered (including within a nested compound statement), or until the end of the compound statement; at the end of a compound statement the state for the pragma is restored to its condition just before the compound statement. If this pragma is used in any other context, the behavior is undefined. If part of an application tests floating-point status flags, sets floating-point control modes, or runs under non-default mode settings, but was translated with the state for the FENV_ACCESS pragma off, the behavior is undefined. The default state (on or off) for the pragma is implementation-defined. (When execution passes from a part of the application translated with FENV_ACCESS off to a part translated with FENV_ACCESS on, the state of the floating-point status flags is unspecified and the floating-point control modes have their default settings.)

APPLICATION USAGE

This header is designed to support the floating-point exception status flags and directed-rounding control modes required by the IEC 60559: 1989 standard, and other similar floating-point state information. Also it is designed to facilitate code portability among all systems.

Certain application programming conventions support the intended model of use for the floating-point environment:

- A function call does not alter its caller's floating-point control modes, clear its caller's floating-point status flags, nor depend on the state of its caller's floating-point status flags unless the function is so documented.
- A function call is assumed to require default floating-point control modes, unless its documentation promises otherwise.

- A function call is assumed to have the potential for raising floating-point exceptions, unless its documentation promises otherwise.

With these conventions, an application can safely assume default floating-point control modes (or be unaware of them). The responsibilities associated with accessing the floating-point environment fall on the application that does so explicitly.

Even though the rounding direction macros may expand to constants corresponding to the values of FLT_ROUNDS, they are not required to do so.

For example:

```
#include <fenv.h>
void f(double x)
{
    #pragma STDC FENV_ACCESS ON
    void g(double);
    void h(double);
    /* ... */
    g(x + 1);
    h(x + 1);
    /* ... */
}
```

If the function *g*() might depend on status flags set as a side-effect of the first *x*+1, or if the second *x*+1 might depend on control modes set as a side-effect of the call to function *g*(), then the application shall contain an appropriately placed invocation as follows:

```
#pragma STDC FENV_ACCESS ON
```

RATIONALE

The fexcept_t Type

fexcept_t does not have to be an integer type. Its values must be obtained by a call to *fegetexceptflag*(), and cannot be created by logical operations from the exception macros. An implementation might simply implement **fexcept_t** as an **int** and use the representations reflected by the exception macros, but is not required to; other representations might contain extra information about the exceptions. **fexcept_t** might be a **struct** with a member for each exception (that might hold the address of the first or last floating-point instruction that caused that exception). The ISO C standard makes no claims about the internals of an **fexcept_t**, and so the user cannot inspect it.

Exception and Rounding Macros

Macros corresponding to unsupported modes and rounding directions are not defined by the implementation and must not be defined by the application. An application might use **#ifdef** to test for this.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH *feclearexcept*(), *fegetenv*(), *fegetexceptflag*(), *fegetround*(), *feholdexcept*(), *feraiseexcept*(), *fetestexcept*(), *feupdateenv*()

8941 **CHANGE HISTORY**

8942 First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

8943 The return types for *feclearexcept*( ), *fegetexceptflag*( ), *feraiseexcept*( ), *fesetexceptflag*( ), *fegetenv*( ),
8944 *fesetenv*( ), and *feupdateenv*( ) are changed from **void** to **int** for alignment with the
8945 ISO/IEC 9899: 1999 standard, Defect Report 202.

8946 **Issue 7**

8947 ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #37 (SD5-XBD-ERN-49) is applied.

8948 ISO/IEC 9899: 1999 standard, Technical Corrigendum 3 #36 is applied.

8949 SD5-XBD-ERN-48 and SD5-XBD-ERN-69 are applied.

8950 This reference page is clarified with respect to macros and symbolic constants.

NAME

float.h — floating types

SYNOPSIS

```
#include <float.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The characteristics of floating types are defined in terms of a model that describes a representation of floating-point numbers and values that provide information about an implementation's floating-point arithmetic.

The following parameters are used to define the model for each floating-point type:

s Sign (±1).

b Base or radix of exponent representation (an integer >1).

e Exponent (an integer between a minimum $e_{\min}$ and a maximum $e_{\max}$).

p Precision (the number of base–*b* digits in the significand).

f_k Non-negative integers less than *b* (the significand digits).

A floating-point number *x* is defined by the following model:

$$x = sb^e \sum_{k=1}^{p} f_k \, b^{-k}, \; e_{\min} \le e \le e_{\max}$$

In addition to normalized floating-point numbers (f_1>0 if x≠0), floating types may be able to contain other kinds of floating-point numbers, such as subnormal floating-point numbers (x≠0, e=$e_{\min}$, f_1=0) and unnormalized floating-point numbers (x≠0, e>$e_{\min}$, f_1=0), and values that are not floating-point numbers, such as infinities and NaNs. A *NaN* is an encoding signifying Not-a-Number. A *quiet NaN* propagates through almost every arithmetic operation without raising a floating-point exception; a *signaling NaN* generally raises a floating-point exception when occurring as an arithmetic operand.

An implementation may give zero and non-numeric values, such as infinities and NaNs, a sign, or may leave them unsigned. Wherever such values are unsigned, any requirement in POSIX.1-2024 to retrieve the sign shall produce an unspecified sign and any requirement to set the sign shall be ignored.

The accuracy of the floating-point operations (`'+'`, `'−'`, `'*'`, `'/'`) and of the functions in **<math.h>** and **<complex.h>** that return floating-point results is implementation-defined, as is the accuracy of the conversion between floating-point internal representations and string representations performed by the functions in **<stdio.h>**, **<stdlib.h>**, and **<wchar.h>**. The implementation may state that the accuracy is unknown.

All integer values in the **<float.h>** header, except FLT_ROUNDS, shall be constant expressions suitable for use in **#if** preprocessing directives; all floating values shall be constant expressions. All except DECIMAL_DIG, FLT_EVAL_METHOD, FLT_RADIX, and FLT_ROUNDS have separate names for all three floating-point types. The floating-point model representation is provided for all values except FLT_EVAL_METHOD and FLT_ROUNDS.

The rounding mode for floating-point addition is characterized by the implementation-defined

value of FLT_ROUNDS:

–1 Indeterminable.

0 Toward zero.

1 To nearest.

2 Toward positive infinity.

3 Toward negative infinity.

All other values for FLT_ROUNDS characterize implementation-defined rounding behavior.

The values of operations with floating operands and values subject to the usual arithmetic conversions and of floating constants are evaluated to a format whose range and precision may be greater than required by the type. The use of evaluation formats is characterized by the implementation-defined value of FLT_EVAL_METHOD:

–1 Indeterminable.

0 Evaluate all operations and constants just to the range and precision of the type.

1 Evaluate operations and constants of type **float** and **double** to the range and precision of the **double** type; evaluate **long double** operations and constants to the range and precision of the **long double** type.

2 Evaluate all operations and constants to the range and precision of the **long double** type.

All other negative values for FLT_EVAL_METHOD characterize implementation-defined behavior.

The presence or absence of subnormal numbers is characterized by the implementation-defined values of FLT_HAS_SUBNORM, DBL_HAS_SUBNORM, and LDBL_HAS_SUBNORM:

–1 Indeterminable.

0 Absent (type does not support subnormal numbers).

1 Present (type does support subnormal numbers).

Note: Characterization as indeterminable is intended if floating-point operations do not consistently interpret subnormal representations as zero, nor as non-zero. Characterization as absent is intended if no floating-point operations produce subnormal results from non-subnormal inputs, even if the type format includes representations of subnormal numbers.

The **<float.h>** header shall define the following values as constant expressions with implementation-defined values that are greater or equal in magnitude (absolute value) to those shown, with the same sign.

- Radix of exponent representation, *b*.

  FLT_RADIX 2

- Number of base-FLT_RADIX digits in the floating-point significand, *p*.

  FLT_MANT_DIG

  DBL_MANT_DIG

  LDBL_MANT_DIG

- Number of decimal digits, *n*, such that any floating-point number with *p* radix *b* digits can be rounded to a floating-point number with *n* decimal digits and back again without change to the value.

$$\begin{cases} p \log_{10} b & \text{if } b \textit{ is a power of } 10 \\ \left\lceil 1 + p \log_{10} b \right\rceil & \textit{otherwise} \end{cases}$$

FLT_DECIMAL_DIG 6

DBL_DECIMAL_DIG 10

LDBL_DECIMAL_DIG 10

- Number of decimal digits, n, such that any floating-point number in the widest supported floating type with $p_{\max}$ radix b digits can be rounded to a floating-point number with n decimal digits and back again without change to the value.

$$\begin{cases} p_{\max} \log_{10} b & \text{if } b \textit{ is a power of } 10 \\ \left\lceil 1 + p_{\max} \log_{10} b \right\rceil & \textit{otherwise} \end{cases}$$

DECIMAL_DIG 10

- Number of decimal digits, q, such that any floating-point number with q decimal digits can be rounded into a floating-point number with p radix b digits and back again without change to the q decimal digits.

$$\begin{cases} p \log_{10} b & \text{if } b \textit{ is a power of } 10 \\ \left\lfloor (p-1) \log_{10} b \right\rfloor & \textit{otherwise} \end{cases}$$

FLT_DIG 6

DBL_DIG 10

LDBL_DIG 10

- Minimum negative integer such that FLT_RADIX raised to that power minus 1 is a normalized floating-point number, $e_{\min}$.

FLT_MIN_EXP

DBL_MIN_EXP

LDBL_MIN_EXP

- Minimum negative integer such that 10 raised to that power is in the range of normalized floating-point numbers.

$$\left\lceil \log_{10} b^{e_{\min}-1} \right\rceil$$

9051 FLT_MIN_10_EXP –37

9052 DBL_MIN_10_EXP –37

9053 LDBL_MIN_10_EXP –37

9054 • Maximum integer such that FLT_RADIX raised to that power minus 1 is a representable
9055 finite floating-point number, e_{max}.

9056 FLT_MAX_EXP

9057 DBL_MAX_EXP

9058 LDBL_MAX_EXP

9059 CX Additionally, FLT_MAX_EXP shall be at least as large as FLT_MANT_DIG,
9060 DBL_MAX_EXP shall be at least as large as DBL_MANT_DIG, and LDBL_MAX_EXP shall
9061 be at least as large as LDBL_MANT_DIG; which has the effect that FLT_MAX, DBL_MAX,
9062 and LDBL_MAX are integral.

9063 • Maximum integer such that 10 raised to that power is in the range of representable finite
9064 floating-point numbers.

$$\left\lfloor \log_{10}((1 - b^{-p})\, b^{e_{max}}) \right\rfloor$$

9065 FLT_MAX_10_EXP +37

9066 DBL_MAX_10_EXP +37

9067 LDBL_MAX_10_EXP +37

9068 The **<float.h>** header shall define the following values as constant expressions with
9069 implementation-defined values that are greater than or equal to those shown:

9070 • Maximum representable finite floating-point number.

$$(1 - b^{-p})\, b^{e_{max}}$$

9071 FLT_MAX 1E+37

9072 DBL_MAX 1E+37

9073 LDBL_MAX 1E+37

9074 The **<float.h>** header shall define the following values as constant expressions with
9075 implementation-defined (positive) values that are less than or equal to those shown:

9076 • The difference between 1 and the least value greater than 1 that is representable in the
9077 given floating-point type, b^{1-p}.

9078 FLT_EPSILON 1E–5

9079 DBL_EPSILON 1E–9

9080 LDBL_EPSILON 1E–9

9081 • Minimum normalized positive floating-point number, $b^{e_{min}-1}$.

FLT_MIN	1E–37
DBL_MIN	1E–37
LDBL_MIN	1E–37

- Minimum positive floating-point number.

FLT_TRUE_MIN	1E–37
DBL_TRUE_MIN	1E–37
LDBL_TRUE_MIN	1E–37

Note: If the presence or absence of subnormal numbers is indeterminable, then the value is intended to be a positive number no greater than the minimum normalized positive number for the type.

APPLICATION USAGE

None.

RATIONALE

All known hardware floating-point formats satisfy the property that the exponent range is larger than the number of digits in the significand. The ISO C standard permits a floating-point format where this property is not true, such that the largest finite value would not be integral; however, it is unlikely that there will ever be hardware support for such a floating-point format, and it introduces boundary cases that portable programs should not have to be concerned with (for example, a non-integral DBL_MAX means that *ceil*( ) would have to worry about overflow). Therefore, this standard imposes an additional requirement that the largest representable finite value is integral.

FUTURE DIRECTIONS

The formula for calculating FLT_MAX, DBL_MAX, and LDBL_MAX is expected to change in the next revision of the ISO C standard such that it only applies if the values are normalized.

SEE ALSO

<complex.h>, **<math.h>**, **<stdio.h>**, **<stdlib.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ISO C standard.

Issue 6

The description of the operations with floating-point values is updated for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #4 (SD5-XBD-ERN-50) and #5 (SD5-XBD-ERN-51) are applied.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0046 [346] and XBD/TC1-2008/0047 [346] are applied.

Issue 8

Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899: 2018 standard.

Austin Group Defect 1752 is applied, changing ``the number of mantissa digits'' to ``the number of digits in the significand''.

Austin Group Defect 1754 is applied, changing the FUTURE DIRECTIONS section.

NAME

fmtmsg.h — message display structures

SYNOPSIS

XSI

```
#include <fmtmsg.h>
```

DESCRIPTION

The **<fmtmsg.h>** header shall define the following symbolic constants:

MM_HARD	Source of the condition is hardware.
MM_SOFT	Source of the condition is software.
MM_FIRM	Source of the condition is firmware.
MM_APPL	Condition detected by application.
MM_UTIL	Condition detected by utility.
MM_OPSYS	Condition detected by operating system.
MM_RECOVER	Recoverable error.
MM_NRECOV	Non-recoverable error.
MM_HALT	Error causing application to halt.
MM_ERROR	Application has encountered a non-fatal fault.
MM_WARNING	Application has detected unusual non-error condition.
MM_INFO	Informative message.
MM_NOSEV	No severity level provided for the message.
MM_PRINT	Display message on standard error.
MM_CONSOLE	Display message on system console.

The table below indicates the null values and identifiers for *fmtmsg*() arguments. The **<fmtmsg.h>** header shall define the symbolic constants in the **Identifier** column, which shall have the type indicated in the **Type** column:

Argument	Type	Null-Value	Identifier
label	**char ***	(**char***)0	MM_NULLLBL
severity	**int**	0	MM_NULLSEV
class	**long**	**0L**	MM_NULLMC
text	**char ***	(**char***)0	MM_NULLTXT
action	**char ***	(**char***)0	MM_NULLACT
tag	**char ***	(**char***)0	MM_NULLTAG

The **<fmtmsg.h>** header shall also define the following symbolic constants for use as return values for *fmtmsg*():

MM_OK	The function succeeded.
MM_NOTOK	The function failed completely.
MM_NOMSG	The function was unable to generate a message on standard error, but otherwise succeeded.

MM_NOCON The function was unable to generate a console message, but otherwise succeeded.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
int fmtmsg(long, const char *, int,
    const char *, const char *, const char *);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH *fmtmsg( )*

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 7

This reference page is clarified with respect to macros and symbolic constants.

NAME

fnmatch.h — filename-matching types

SYNOPSIS

```
#include <fnmatch.h>
```

DESCRIPTION

The **<fnmatch.h>** header shall define the following symbolic constants:

FNM_NOMATCH	The string does not match the specified pattern.
FNM_PATHNAME	<slash> in *string* only matches <slash> in *pattern*.
FNM_PERIOD	Leading <period> in *string* only matches <period> in *pattern*.
FNM_NOESCAPE	Disable backslash escaping.
FNM_CASEFOLD	Compare *string* and *pattern* in a case-insensitive manner. See Section 4.1 (on page 95).
FNM_IGNORECASE	Equivalent to FNM_CASEFOLD.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
int fnmatch(const char *, const char *, int);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH *fnmatch( )*

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 6

The FNM_NOSYS constant is marked obsolescent.

Issue 7

The obsolescent FNM_NOSYS constant is removed.

This reference page is clarified with respect to macros and symbolic constants.

Issue 8

Austin Group Defect 1031 is applied, adding FNM_CASEFOLD and FNM_IGNORECASE.

The description of FNM_PERIOD is updated to eliminate the use of ``must''.

NAME

ftw.h — file tree traversal

SYNOPSIS

XSI

```
#include <ftw.h>
```

DESCRIPTION

The **<ftw.h>** header shall define the **FTW** structure, which shall include at least the following members:

```
int  base
int  level
```

The **<ftw.h>** header shall define the following symbolic constants for use as values of the third argument to the application-supplied function that is passed as the second argument to *nftw*( ):

FTW_F	Non-directory file.
FTW_D	Directory.
FTW_DNR	Directory without read permission.
FTW_DP	Directory with subdirectories visited.
FTW_NS	Unknown type; *stat*( ) failed.
FTW_SL	Symbolic link.
FTW_SLN	Symbolic link that names a nonexistent file.

The **<ftw.h>** header shall define the following symbolic constants for use as values of the fourth argument to *nftw*( ):

FTW_PHYS	Physical walk, does not follow symbolic links. Otherwise, *nftw*( ) follows links but does not walk down any path that crosses itself.
FTW_MOUNT	The walk only reports files that have the same device ID as the starting directory and does not descend below directories that have a different device ID than the starting directory.
FTW_XDEV	The walk does not descend below directories that have a different device ID than the starting directory.
FTW_DEPTH	All subdirectories are visited before the directory itself.
FTW_CHDIR	The walk changes to each directory before reading it.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
int nftw(const char *, int (*)(const char *, const struct stat *,
    int, struct FTW *), int, int);
```

The **<ftw.h>** header shall define the **stat** structure and the symbolic names for *st_mode* and the file type test macros as described in **<sys/stat.h>**.

Inclusion of the **<ftw.h>** header may also make visible all symbols from **<sys/stat.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/stat.h>

XSH *nftw*( )

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

A description of FTW_DP is added.

Issue 7

The *ftw*( ) function is marked obsolescent.

This reference page is clarified with respect to macros and symbolic constants.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0048 [403] is applied.

Issue 8

Austin Group Defect 1133 is applied, adding FTW_XDEV.

Austin Group Defect 1210 is applied, changing the description of FTW_MOUNT.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

NAME

glob.h — pathname pattern-matching types

SYNOPSIS

```
#include <glob.h>
```

DESCRIPTION

The **<glob.h>** header shall define the structures and symbolic constants used by the *glob*( ) function.

The **<glob.h>** header shall define the **glob_t** structure type, which shall include at least the following members:

```
size_t   gl_pathc  Count of paths matched by pattern.
char    **gl_pathv  Pointer to a list of matched pathnames.
size_t   gl_offs   Slots to reserve at the beginning of gl_pathv.
```

The **<glob.h>** header shall define the **size_t** type as described in **<sys/types.h>**.

The **<glob.h>** header shall define the following symbolic constants as values for the *flags* argument:

GLOB_APPEND	Append generated pathnames to those previously obtained.
GLOB_DOOFFS	Specify how many null pointers to add to the beginning of *gl_pathv*.
GLOB_ERR	Cause *glob*( ) to return on error.
GLOB_MARK	Each pathname that is a directory that matches *pattern* has a <slash> appended.
GLOB_NOCHECK	If *pattern* does not match any pathname, then return a list consisting of only *pattern*.
GLOB_NOESCAPE	Disable backslash escaping.
GLOB_NOSORT	Do not sort the pathnames returned.

The **<glob.h>** header shall define the following symbolic constants as error return values:

GLOB_ABORTED	The scan was stopped because GLOB_ERR was set or (**errfunc*)( ) returned non-zero.
GLOB_NOMATCH	The pattern does not match any existing pathname, and GLOB_NOCHECK was not set in *flags*.
GLOB_NOSPACE	An attempt to allocate memory failed.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int  glob(const char *restrict, int, int(*)(const char *, int),
         glob_t *restrict);
void globfree(glob_t *);
```


APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>

XSH *glob*( )

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 6

The **restrict** keyword is added to the prototype for *glob*( ).

The GLOB_NOSYS constant is marked obsolescent.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/8 is applied, correcting the *glob*( ) prototype definition by removing the **restrict** qualifier from the function pointer argument.

Issue 7

SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t**

The obsolescent GLOB_NOSYS constant is removed.

This reference page is clarified with respect to macros and symbolic constants.

NAME

grp.h — group structure

SYNOPSIS

```
#include <grp.h>
```

DESCRIPTION

The **<grp.h>** header shall declare the **group** structure, which shall include the following members:

`char *gr_name`	The name of the group.
`gid_t gr_gid`	Numerical group ID.
`char **gr_mem`	Pointer to a null-terminated array of character pointers to member names.

The **<grp.h>** header shall define the **gid_t** and **size_t** types as described in **<sys/types.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

XSI
```
void           endgrent(void);
struct group  *getgrent(void);
struct group  *getgrgid(gid_t);
int            getgrgid_r(gid_t, struct group *, char *,
                   size_t, struct group **);
struct group  *getgrnam(const char *);
int            getgrnam_r(const char *, struct group *, char *,
                   size_t , struct group **);
```
XSI
```
void           setgrent(void);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>

XSH *endgrent*( ), *getgrgid*( ), *getgrnam*( )

CHANGE HISTORY

First released in Issue 1.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The definition of **gid_t** is mandated.
- The *getgrgid_r*( ) and *getgrnam_r*( ) functions are marked as part of the Thread-Safe Functions option.

Issue 7

SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0049 [24] is applied.

NAME

iconv.h — codeset conversion facility

SYNOPSIS

```
#include <iconv.h>
```

DESCRIPTION

The **<iconv.h>** header shall define the following types:

iconv_t Identifies the conversion from one codeset to another.

size_t As described in **<sys/types.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
size_t  iconv(iconv_t, char **restrict, size_t *restrict,
            char **restrict, size_t *restrict);
int     iconv_close(iconv_t);
iconv_t iconv_open(const char *, const char *);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>

XSH *iconv*( ), *iconv_close*( ), *iconv_open*( )

CHANGE HISTORY

First released in Issue 4.

Issue 6

The **restrict** keyword is added to the prototype for *iconv*( ).

Issue 7

SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

The **<iconv.h>** header is moved from the XSI option to the Base.

NAME

inttypes.h — fixed size integer types

SYNOPSIS

```
#include <inttypes.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to enable the visibility of these symbols in this header.

The **<inttypes.h>** header shall include the **<stdint.h>** header.

The **<inttypes.h>** header shall define at least the following types:

imaxdiv_t Structure type that is the type of the value returned by the *imaxdiv*( ) function.

CX **wchar_t** As described in **<stddef.h>**.

The **<inttypes.h>** header shall define the following macros. Each expands to a character string literal containing a conversion specifier, possibly modified by a length modifier, suitable for use within the *format* argument of a formatted input/output function when converting the corresponding integer type. These macros have the general form of PRI (character string literals for the *fprintf*( ) and *fwprintf*( ) family of functions) or SCN (character string literals for the *fscanf*( ) and *fwscanf*( ) family of functions), followed by the conversion specifier, followed by a name corresponding to a similar type name in **<stdint.h>**. In these names, *N* represents the width of the type as described in **<stdint.h>**. For example, *PRIdFAST32* can be used in a format string to print the value of an integer of type **int_fast32_t**.

The *fprintf*( ) macros for signed integers are:

PRId*N*	PRIdLEAST*N*	PRIdFAST*N*	PRIdMAX	PRIdPTR
PRIi*N*	PRIiLEAST*N*	PRIiFAST*N*	PRIiMAX	PRIiPTR

The *fprintf*( ) macros for unsigned integers are:

PRIo*N*	PRIoLEAST*N*	PRIoFAST*N*	PRIoMAX	PRIoPTR
PRIu*N*	PRIuLEAST*N*	PRIuFAST*N*	PRIuMAX	PRIuPTR
PRIx*N*	PRIxLEAST*N*	PRIxFAST*N*	PRIxMAX	PRIxPTR
PRIX*N*	PRIXLEAST*N*	PRIXFAST*N*	PRIXMAX	PRIXPTR

The *fscanf*( ) macros for signed integers are:

SCNd*N*	SCNdLEAST*N*	SCNdFAST*N*	SCNdMAX	SCNdPTR
SCNi*N*	SCNiLEAST*N*	SCNiFAST*N*	SCNiMAX	SCNiPTR

The *fscanf*( ) macros for unsigned integers are:

SCNo*N*	SCNoLEAST*N*	SCNoFAST*N*	SCNoMAX	SCNoPTR
SCNu*N*	SCNuLEAST*N*	SCNuFAST*N*	SCNuMAX	SCNuPTR
SCNx*N*	SCNxLEAST*N*	SCNxFAST*N*	SCNxMAX	SCNxPTR

For each type that the implementation provides in **<stdint.h>**, the corresponding *fprintf*( ) and *fwprintf*( ) macros shall be defined and the corresponding *fscanf*( ) and *fwscanf*( ) macros shall be defined unless the implementation does not have a suitable modifier for the type.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.


```
intmax_t  imaxabs(intmax_t);
imaxdiv_t imaxdiv(intmax_t, intmax_t);
intmax_t  strtoimax(const char *restrict, char **restrict, int);
uintmax_t strtoumax(const char *restrict, char **restrict, int);
intmax_t  wcstoimax(const wchar_t *restrict, wchar_t **restrict, int);
uintmax_t wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);
```

EXAMPLES

```
#include <inttypes.h>
#include <wchar.h>
int main(void)
{
    uintmax_t i = UINTMAX_MAX; // This type always exists.
    wprintf(L"The largest integer value is %020"
        PRIxMAX "\n", i);
    return 0;
}
```

APPLICATION USAGE

The purpose of **<inttypes.h>** is to provide a set of integer types whose definitions are consistent across machines and independent of operating systems and other implementation idiosyncrasies. It defines, through **typedef**, integer types of various sizes. Implementations are free to **typedef** them as ISO C standard integer types or extensions that they support. Consistent use of this header will greatly increase the portability of applications across platforms.

RATIONALE

The ISO/IEC 9899:1990 standard specified that the language should support four signed and unsigned integer data types—**char**, **short**, **int**, and **long**—but placed very little requirement on their size other than that **int** and **short** be at least 16 bits and **long** be at least as long as **int** and not smaller than 32 bits. For 16-bit systems, most implementations assigned 8, 16, 16, and 32 bits to **char**, **short**, **int**, and **long**, respectively. For 32-bit systems, the common practice has been to assign 8, 16, 32, and 32 bits to these types. This difference in **int** size can create some problems for users who migrate from one system to another which assigns different sizes to integer types, because the ISO C standard integer promotion rule can produce silent changes unexpectedly. The need for defining an extended integer type increased with the introduction of 64-bit systems.

FUTURE DIRECTIONS

Macro names beginning with PRI or SCN followed by any lowercase letter or `'X'` may be added to the macros defined in the **<inttypes.h>** header.

SEE ALSO

<stddef.h>

XSH Section 2.2 (on page 496), *imaxabs*( ), *imaxdiv*( ), *strtoimax*( ), *wcstoimax*( )

CHANGE HISTORY

First released in Issue 5.

Issue 6

The Open Group Base Resolution bwg97-006 is applied.

This reference page is updated to align with the ISO/IEC 9899:1999 standard.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0050 [211] is applied.

NAME

iso646.h — alternative spellings

SYNOPSIS

```
#include <iso646.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The **<iso646.h>** header shall define the following eleven macros (on the left) that expand to the corresponding tokens (on the right):

and	`&&`
and_eq	`&=`
bitand	`&`
bitor	`|`
compl	`~`
not	`!`
not_eq	`!=`
or	`||`
or_eq	`|=`
xor	`^`
xor_eq	`^=`

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

None.

CHANGE HISTORY

First released in Issue 5. Derived from ISO/IEC 9899: 1990/Amendment 1: 1995 (E).

NAME

langinfo.h — language information constants

SYNOPSIS

```
#include <langinfo.h>
```

DESCRIPTION

The **<langinfo.h>** header shall define the symbolic constants used to identify items of *langinfo* data (see *nl_langinfo*( )).

The **<langinfo.h>** header shall define the **locale_t** type as described in **<locale.h>**.

The **<langinfo.h>** header shall define the **nl_item** type as described in **<nl_types.h>**.

The **<langinfo.h>** header shall define the following symbolic constants with type **nl_item**. The entries under **Category** indicate in which *setlocale*( ) category each item is defined.

Constant	Category	Meaning
CODESET	*LC_CTYPE*	Codeset name.
D_T_FMT	*LC_TIME*	String for formatting date and time.
D_FMT	*LC_TIME*	Date format string.
T_FMT	*LC_TIME*	Time format string.
T_FMT_AMPM	*LC_TIME*	Time format string using 12-hour clock format, if supported in the locale; if the 12-hour format is not supported, this shall be either an empty string or a string specifying a 24-hour clock format.
AM_STR	*LC_TIME*	Ante-meridiem affix; if AM_STR and PM_STR are both empty strings, the 12-hour format is not supported in the locale.
PM_STR	*LC_TIME*	Post-meridiem affix; if AM_STR and PM_STR are both empty strings, the 12-hour format is not supported in the locale.
DAY_1	*LC_TIME*	Name of the first day of the week (for example, Sunday).
DAY_2	*LC_TIME*	Name of the second day of the week (for example, Monday).
DAY_3	*LC_TIME*	Name of the third day of the week (for example, Tuesday).
DAY_4	*LC_TIME*	Name of the fourth day of the week (for example, Wednesday).
DAY_5	*LC_TIME*	Name of the fifth day of the week (for example, Thursday).
DAY_6	*LC_TIME*	Name of the sixth day of the week (for example, Friday).
DAY_7	*LC_TIME*	Name of the seventh day of the week (for example, Saturday).
ABDAY_1	*LC_TIME*	Abbreviated name of the first day of the week.
ABDAY_2	*LC_TIME*	Abbreviated name of the second day of the week.
ABDAY_3	*LC_TIME*	Abbreviated name of the third day of the week.
ABDAY_4	*LC_TIME*	Abbreviated name of the fourth day of the week.
ABDAY_5	*LC_TIME*	Abbreviated name of the fifth day of the week.
ABDAY_6	*LC_TIME*	Abbreviated name of the sixth day of the week.
ABDAY_7	*LC_TIME*	Abbreviated name of the seventh day of the week.
MON_1	*LC_TIME*	Name of the first month of the year.
MON_2	*LC_TIME*	Name of the second month.
MON_3	*LC_TIME*	Name of the third month.
MON_4	*LC_TIME*	Name of the fourth month.
MON_5	*LC_TIME*	Name of the fifth month.
MON_6	*LC_TIME*	Name of the sixth month.

Constant	Category	Meaning
MON_7	*LC_TIME*	Name of the seventh month.
MON_8	*LC_TIME*	Name of the eighth month.
MON_9	*LC_TIME*	Name of the ninth month.
MON_10	*LC_TIME*	Name of the tenth month.
MON_11	*LC_TIME*	Name of the eleventh month.
MON_12	*LC_TIME*	Name of the twelfth month.
ALTMON_1	*LC_TIME*	Name of the alternative appropriate first month of the year.
ALTMON_2	*LC_TIME*	Name of the alternative appropriate second month.
ALTMON_3	*LC_TIME*	Name of the alternative appropriate third month.
ALTMON_4	*LC_TIME*	Name of the alternative appropriate fourth month.
ALTMON_5	*LC_TIME*	Name of the alternative appropriate fifth month.
ALTMON_6	*LC_TIME*	Name of the alternative appropriate sixth month.
ALTMON_7	*LC_TIME*	Name of the alternative appropriate seventh month.
ALTMON_8	*LC_TIME*	Name of the alternative appropriate eighth month.
ALTMON_9	*LC_TIME*	Name of the alternative appropriate ninth month.
ALTMON_10	*LC_TIME*	Name of the alternative appropriate tenth month.
ALTMON_11	*LC_TIME*	Name of the alternative appropriate eleventh month.
ALTMON_12	*LC_TIME*	Name of the alternative appropriate twelfth month.
ABMON_1	*LC_TIME*	Abbreviated name of the first month.
ABMON_2	*LC_TIME*	Abbreviated name of the second month.
ABMON_3	*LC_TIME*	Abbreviated name of the third month.
ABMON_4	*LC_TIME*	Abbreviated name of the fourth month.
ABMON_5	*LC_TIME*	Abbreviated name of the fifth month.
ABMON_6	*LC_TIME*	Abbreviated name of the sixth month.
ABMON_7	*LC_TIME*	Abbreviated name of the seventh month.
ABMON_8	*LC_TIME*	Abbreviated name of the eighth month.
ABMON_9	*LC_TIME*	Abbreviated name of the ninth month.
ABMON_10	*LC_TIME*	Abbreviated name of the tenth month.
ABMON_11	*LC_TIME*	Abbreviated name of the eleventh month.
ABMON_12	*LC_TIME*	Abbreviated name of the twelfth month.
ABALTMON_1	*LC_TIME*	Abbreviated alternative name of the first month of the year.
ABALTMON_2	*LC_TIME*	Abbreviated alternative name of the second month.
ABALTMON_3	*LC_TIME*	Abbreviated alternative name of the third month.
ABALTMON_4	*LC_TIME*	Abbreviated alternative name of the fourth month.
ABALTMON_5	*LC_TIME*	Abbreviated alternative name of the fifth month.
ABALTMON_6	*LC_TIME*	Abbreviated alternative name of the sixth month.
ABALTMON_7	*LC_TIME*	Abbreviated alternative name of the seventh month.
ABALTMON_8	*LC_TIME*	Abbreviated alternative name of the eighth month.
ABALTMON_9	*LC_TIME*	Abbreviated alternative name of the ninth month.
ABALTMON_10	*LC_TIME*	Abbreviated alternative name of the tenth month.
ABALTMON_11	*LC_TIME*	Abbreviated alternative name of the eleventh month.
ABALTMON_12	*LC_TIME*	Abbreviated alternative name of the twelfth month.
ERA	*LC_TIME*	Era description segments.
ERA_D_FMT	*LC_TIME*	Era date format string.
ERA_D_T_FMT	*LC_TIME*	Era date and time format string.
ERA_T_FMT	*LC_TIME*	Era time format string.
ALT_DIGITS	*LC_TIME*	Alternative symbols for digits.
RADIXCHAR	*LC_NUMERIC*	Radix character.
THOUSEP	*LC_NUMERIC*	Separator for thousands.
YESEXPR	*LC_MESSAGES*	Affirmative response expression.

Constant	Category	Meaning
NOEXPR	*LC_MESSAGES*	Negative response expression.
CRNCYSTR	*LC_MONETARY*	Local currency symbol, preceded by `'-'` if the symbol should appear before the value, `'+'` if the symbol should appear after the value, or `'.'` if the symbol should replace the radix character. If the local currency symbol is the empty string, implementations may return the empty string (`""`).

If the locale's values for **p_cs_precedes** and **n_cs_precedes** do not match, the value of *nl_langinfo*(CRNCYSTR) and *nl_langinfo_l*(CRNCYSTR,loc) is unspecified.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
char *nl_langinfo(nl_item);
char *nl_langinfo_l(nl_item, locale_t);
```

Inclusion of the **<langinfo.h>** header may also make visible all symbols from **<nl_types.h>**.

APPLICATION USAGE

Wherever possible, users are advised to use functions compatible with those in the ISO C standard to access items of *langinfo* data. In particular, the *strftime*( ) function should be used to access date and time information defined in category *LC_TIME*. The *localeconv*( ) function should be used to access information corresponding to RADIXCHAR, THOUSEP, and CRNCYSTR.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 7 (on page 127), **<locale.h>**, **<nl_types.h>**

XSH *nl_langinfo*( ), *localeconv*( ), *strfmon*( ), *strftime*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

The constants YESSTR and NOSTR are marked LEGACY.

Issue 6

The constants YESSTR and NOSTR are removed.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/9 is applied, adding a sentence to the``Meaning'' column entry for the CRNCYSTR constant. This change is to accommodate historic practice.

Issue 7

The **<langinfo.h>** header is moved from the XSI option to the Base.

The *nl_langinfo_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

This reference page is clarified with respect to macros and symbolic constants, and a declaration for the **locale_t** type is added.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0051 [107] is applied.

Issue 8

Austin Group Defects 258 and 1166 are applied, adding the ALTMON_*x* and ABALTMON_*x* symbolic constants.

Austin Group Defect 1307 is applied, changing the AM_STR, PM_STR, and T_FMT_AMPM constants in relation to locales that do not support the 12-hour clock format.

NAME

libgen.h — definitions for pattern matching functions

SYNOPSIS

XSI
```
#include <libgen.h>
```

DESCRIPTION

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
char  *basename(char *);
char  *dirname(char *);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH *basename*( ), *dirname*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

The function prototypes for *basename*( ) and *dirname*( ) are changed to indicate that the first argument is of type **char *** rather than **const char ***.

Issue 6

The **__loc1** symbol and the *regcmp*( ) and *regex*( ) functions are removed.

NAME

libintl.h — international messaging

SYNOPSIS

```
#include <libintl.h>
```

DESCRIPTION

The **<libintl.h>** header may define the macro TEXTDOMAINMAX. If defined, it shall have the same value as {TEXTDOMAIN_MAX} in **<limits.h>**.

The **<libintl.h>** header shall define the **locale_t** type as described in **<locale.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
char   *bindtextdomain(const char *, const char *);
char   *bind_textdomain_codeset(const char *, const char *);
char   *dcgettext(const char *, const char *, int);
char   *dcgettext_l(const char *, const char *, int, locale_t);
char   *dcngettext(const char *, const char *, const char *,
            unsigned long int, int);
char   *dcngettext_l(const char *, const char *, const char *,
            unsigned long int, int, locale_t);
char   *dgettext(const char *, const char *);
char   *dgettext_l(const char *, const char *, locale_t);
char   *dngettext(const char *, const char *, const char *,
            unsigned long int);
char   *dngettext_l(const char *, const char *, const char *,
            unsigned long int, locale_t);
char   *gettext(const char *);
char   *gettext_l(const char *, locale_t);
char   *ngettext(const char *, const char *, unsigned long int);
char   *ngettext_l(const char *, const char *,
            unsigned long int, locale_t);
char   *textdomain(const char *);
```

APPLICATION USAGE

None.

RATIONALE

Some historical implementations defined TEXTDOMAINMAX in this header. This standard instead defines {TEXTDOMAIN_MAX} in **<limits.h>**. This was done to allow the maximum length of a text domain name to vary depending on the filesystem type used to store message catalogs. Implementations are allowed to continue to define TEXTDOMAINMAX in this header as an extension to the standard (see XSH Section 2.2.2, on page 498).

FUTURE DIRECTIONS

None.

SEE ALSO

<locale.h>

XSH *gettext*, *bindtextdomain*( )

9731 **CHANGE HISTORY**
9732 First released in Issue 8.

NAME

limits.h — implementation-defined constants

SYNOPSIS

```
#include <limits.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to enable the visibility of these symbols in this header.

Many of the symbols listed here are not defined by the ISO C standard. Such symbols are not shown as CX shaded, except under the heading ``Numerical Limits''.

The **<limits.h>** header shall define macros and symbolic constants for various limits. Different categories of limits are described below, representing various limits on resources that the implementation imposes on applications. All macros and symbolic constants defined in this header shall be suitable for use in **#if** preprocessing directives.

Implementations may choose any appropriate value for each limit, provided it is not more restrictive than the Minimum Acceptable Values listed below. Symbolic constant names beginning with _POSIX may be found in **<unistd.h>**.

Applications should not assume any particular value for a limit. To achieve maximum portability, an application should not require more resource than the Minimum Acceptable Value quantity. However, an application wishing to avail itself of the full amount of a resource available on an implementation may make use of the value given in **<limits.h>** on that particular implementation, by using the macros and symbolic constants listed below. It should be noted, however, that many of the listed limits are not invariant, and at runtime, the value of the limit may differ from those given in this header, for the following reasons:

- The limit is pathname-dependent.
- The limit differs between the compile and runtime machines.
- The limit has been changed at runtime by an application (see *setrlimit*( )).

For these reasons, an application can use the *fpathconf*( ), *getrlimit*( ), *pathconf*( ), and *sysconf*( ) functions to determine the actual value of a limit at runtime.

The items in the list ending in _MIN give the most negative values that the mathematical types are guaranteed to be capable of representing. Numbers of a more negative value may be supported on some implementations, as indicated by the **<limits.h>** header on the implementation, but applications requiring such numbers are not guaranteed to be portable to all implementations. For positive constants ending in _MIN, this indicates the minimum acceptable value.

Runtime Invariant Values (Possibly Indeterminate)

A definition of one of the symbolic constants in the following list shall be omitted from **<limits.h>** on specific implementations where the corresponding value is equal to or greater than the stated minimum, but is unspecified.

This indetermination might depend on the amount of available memory space on a specific instance of a specific implementation. The actual value supported by a specific instance shall be provided by the *sysconf*( ) function.

{AIO_LISTIO_MAX}
Maximum number of I/O operations in a single list I/O call supported by the implementation.
Minimum Acceptable Value: {_POSIX_AIO_LISTIO_MAX}

{AIO_MAX}
Maximum number of outstanding asynchronous I/O operations supported by the implementation.
Minimum Acceptable Value: {_POSIX_AIO_MAX}

{AIO_PRIO_DELTA_MAX}
The maximum amount by which a process can decrease its asynchronous I/O priority level from its own scheduling priority.
Minimum Acceptable Value: 0

{ARG_MAX}
Maximum length of argument to the *exec* functions including environment data.
Minimum Acceptable Value: {_POSIX_ARG_MAX}

{ATEXIT_MAX}
Maximum number of functions that can be registered with *atexit*( ) or *at_quick_exit*( ). The limit shall apply independently to each function.
Minimum Acceptable Value: 32

{CHILD_MAX}
Maximum number of simultaneous processes per real user ID.
Minimum Acceptable Value: {_POSIX_CHILD_MAX}

{DELAYTIMER_MAX}
Maximum number of timer expiration overruns.
Minimum Acceptable Value: {_POSIX_DELAYTIMER_MAX}

{HOST_NAME_MAX}
Maximum length of a host name (not including the terminating null) as returned from the *gethostname*( ) function.
Minimum Acceptable Value: {_POSIX_HOST_NAME_MAX}

XSI {IOV_MAX}
Maximum number of **iovec** structures that one process has available for use with *readv*( ) or *writev*( ).
Minimum Acceptable Value: {_XOPEN_IOV_MAX}

{LOGIN_NAME_MAX}
Maximum length of a login name.
Minimum Acceptable Value: {_POSIX_LOGIN_NAME_MAX}

MSG {MQ_OPEN_MAX}
The maximum number of open message queue descriptors a process may hold.
Minimum Acceptable Value: {_POSIX_MQ_OPEN_MAX}

MSG {MQ_PRIO_MAX}
The maximum number of message priorities supported by the implementation.
Minimum Acceptable Value: {_POSIX_MQ_PRIO_MAX}

{OPEN_MAX}
A value one greater than the maximum value that the system may assign to a newly-created file descriptor.
Minimum Acceptable Value: {_POSIX_OPEN_MAX}

{PAGESIZE}
Size in bytes of a page.
Minimum Acceptable Value: 1

XSI {PAGE_SIZE}
Equivalent to {PAGESIZE}. If either {PAGESIZE} or {PAGE_SIZE} is defined, the other is defined with the same value.

{PTHREAD_DESTRUCTOR_ITERATIONS}
Maximum number of attempts made to destroy a thread's thread-specific data values on thread exit.
Minimum Acceptable Value: {_POSIX_THREAD_DESTRUCTOR_ITERATIONS}

{PTHREAD_KEYS_MAX}
Maximum number of data keys that can be created by a process.
Minimum Acceptable Value: {_POSIX_THREAD_KEYS_MAX}

{PTHREAD_STACK_MIN}
Minimum size in bytes of thread stack storage.
Minimum Acceptable Value: 0

{PTHREAD_THREADS_MAX}
Maximum number of threads that can be created per process.
Minimum Acceptable Value: {_POSIX_THREAD_THREADS_MAX}

{RTSIG_MAX}
Maximum number of realtime signals reserved for application use in this implementation.
Minimum Acceptable Value: {_POSIX_RTSIG_MAX}

{SEM_NSEMS_MAX}
Maximum number of semaphores that a process may have.
Minimum Acceptable Value: {_POSIX_SEM_NSEMS_MAX}

{SEM_VALUE_MAX}
The maximum value a semaphore may have.
Minimum Acceptable Value: {_POSIX_SEM_VALUE_MAX}

{SIGQUEUE_MAX}
Maximum number of queued signals that a process may send and have pending at the receiver(s) at any time.
Minimum Acceptable Value: {_POSIX_SIGQUEUE_MAX}

SS | TSP {SS_REPL_MAX}
The maximum number of replenishment operations that may be simultaneously pending for a particular sporadic server scheduler.
Minimum Acceptable Value: {_POSIX_SS_REPL_MAX}

{STREAM_MAX}
Maximum number of streams that one process can have open at one time. If defined, it has the same value as {FOPEN_MAX} (see **<stdio.h>**).
Minimum Acceptable Value: {_POSIX_STREAM_MAX}

{SYMLOOP_MAX}
Maximum number of symbolic links that can be reliably traversed in the resolution of a pathname in the absence of a loop.
Minimum Acceptable Value: {_POSIX_SYMLOOP_MAX}

{TIMER_MAX}
Maximum number of timers per process supported by the implementation.
Minimum Acceptable Value: {_POSIX_TIMER_MAX}

{TTY_NAME_MAX}
Maximum length of terminal device name.
Minimum Acceptable Value: {_POSIX_TTY_NAME_MAX}

{TZNAME_MAX}
Maximum number of bytes supported for the name of a timezone (not of the *TZ* variable).
Minimum Acceptable Value: {_POSIX_TZNAME_MAX}

Note: The length given by {TZNAME_MAX} does not include the quoting characters mentioned in Section 8.3 (on page 174).

Pathname Variable Values

The values in the following list may be constants within an implementation or may vary from one pathname to another. For example, file systems or directories may have different characteristics.

A definition of one of the symbolic constants in the following list shall be omitted from the **<limits.h>** header on specific implementations where the corresponding value is equal to or greater than the stated minimum, but where the value can vary depending on the file to which it is applied. The actual value supported for a specific pathname shall be provided by the *pathconf*( ) function.

{FILESIZEBITS}
Minimum number of bits needed to represent, as a signed integer value, the maximum size of a regular file allowed in the specified directory.
Minimum Acceptable Value: 32

{LINK_MAX}
Maximum number of links to a single file.
Minimum Acceptable Value: {_POSIX_LINK_MAX}

{MAX_CANON}
Maximum number of bytes in a terminal canonical input line.
Minimum Acceptable Value: {_POSIX_MAX_CANON}

{MAX_INPUT}
Minimum number of bytes for which space is available in a terminal input queue; therefore, the maximum number of bytes a conforming application may require to be typed as input before reading them.
Minimum Acceptable Value: {_POSIX_MAX_INPUT}

{NAME_MAX}
Maximum number of bytes in a filename (not including the terminating null of a filename string).
Minimum Acceptable Value: {_POSIX_NAME_MAX}
XSI Minimum Acceptable Value: {_XOPEN_NAME_MAX}

{PATH_MAX}
Maximum number of bytes the implementation stores as a pathname in a user-supplied buffer of unspecified size, including the terminating null character. Minimum number the implementation shall accept as the maximum number of bytes in a pathname.
Minimum Acceptable Value: {_POSIX_PATH_MAX}

9910 XSI Minimum Acceptable Value: {_XOPEN_PATH_MAX}

9911 {PIPE_BUF}
9912 Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
9913 Minimum Acceptable Value: {_POSIX_PIPE_BUF}

9914 ADV {POSIX_ALLOC_SIZE_MIN}
9915 Minimum number of bytes of storage actually allocated for any portion of a file.
9916 Minimum Acceptable Value: Not specified.

9917 ADV {POSIX_REC_INCR_XFER_SIZE}
9918 Recommended increment for file transfer sizes between the
9919 {POSIX_REC_MIN_XFER_SIZE} and {POSIX_REC_MAX_XFER_SIZE} values.
9920 Minimum Acceptable Value: Not specified.

9921 ADV {POSIX_REC_MAX_XFER_SIZE}
9922 Maximum recommended file transfer size.
9923 Minimum Acceptable Value: Not specified.

9924 ADV {POSIX_REC_MIN_XFER_SIZE}
9925 Minimum recommended file transfer size.
9926 Minimum Acceptable Value: Not specified.

9927 ADV {POSIX_REC_XFER_ALIGN}
9928 Recommended file transfer buffer alignment.
9929 Minimum Acceptable Value: Not specified.

9930 {SYMLINK_MAX}
9931 Maximum number of bytes in a symbolic link.
9932 Minimum Acceptable Value: {_POSIX_SYMLINK_MAX}

9933 {TEXTDOMAIN_MAX}
9934 Maximum length of a text domain name, not including the terminating null byte.
9935 Minimum Acceptable Value: {_POSIX_NAME_MAX} – 3
9936 XSI Minimum Acceptable Value: {_XOPEN_NAME_MAX} – 3

9937 **Runtime Increasable Values**

9938 The magnitude limitations in the following list shall be fixed by specific implementations. An
9939 application should assume that the value of the symbolic constant defined by **<limits.h>** in a
9940 specific implementation is the minimum that pertains whenever the application is run under
9941 that implementation. A specific instance of a specific implementation may increase the value
9942 relative to that supplied by **<limits.h>** for that implementation. The actual value supported by a
9943 specific instance shall be provided by the *sysconf*( ) function.

9944 {BC_BASE_MAX}
9945 Maximum *obase* values allowed by the *bc* utility.
9946 Minimum Acceptable Value: {_POSIX2_BC_BASE_MAX}

9947 {BC_DIM_MAX}
9948 Maximum number of elements permitted in an array by the *bc* utility.
9949 Minimum Acceptable Value: {_POSIX2_BC_DIM_MAX}

9950 {BC_SCALE_MAX}
9951 Maximum *scale* value allowed by the *bc* utility.
9952 Minimum Acceptable Value: {_POSIX2_BC_SCALE_MAX}

{BC_STRING_MAX}
Maximum length of a string constant accepted by the *bc* utility.
Minimum Acceptable Value: {_POSIX2_BC_STRING_MAX}

{CHARCLASS_NAME_MAX}
Maximum number of bytes in a character class name.
Minimum Acceptable Value: {_POSIX2_CHARCLASS_NAME_MAX}

{COLL_WEIGHTS_MAX}
Maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order** keyword in the locale definition file; see Chapter 7 (on page 127).
Minimum Acceptable Value: {_POSIX2_COLL_WEIGHTS_MAX}

{EXPR_NEST_MAX}
Maximum number of expressions that can be nested within parentheses by the *expr* utility.
Minimum Acceptable Value: {_POSIX2_EXPR_NEST_MAX}

{LINE_MAX}
Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <newline>.
Minimum Acceptable Value: {_POSIX2_LINE_MAX}

{NGROUPS_MAX}
Maximum number of simultaneous supplementary group IDs per process.
Minimum Acceptable Value: {_POSIX_NGROUPS_MAX}

{RE_DUP_MAX}
Maximum number of repeated occurrences of a BRE or ERE interval expression; see Section 9.3.6 (on page 185) and Section 9.4.6 (on page 189).
Minimum Acceptable Value: {_POSIX_RE_DUP_MAX}

Maximum Values

The **<limits.h>** header shall define the following symbolic constants with the values shown. These are the most restrictive values for certain features on an implementation. A conforming implementation shall provide values no larger than these values. A conforming application shall not require a smaller value for correct operation.

{_POSIX_CLOCKRES_MIN}
The resolution of the CLOCK_REALTIME and CLOCK_MONOTONIC clocks, in nanoseconds.
Value: 20 000 000

Minimum Values

The **<limits.h>** header shall define the following symbolic constants with the values shown. These are the most restrictive values for certain features on an implementation conforming to this volume of POSIX.1-2024. Related symbolic constants are defined elsewhere in this volume of POSIX.1-2024 which reflect the actual implementation and which need not be as restrictive. For each of these limits, a conforming implementation shall provide a value at least this large or shall have no limit. A strictly conforming application shall not require a larger value for correct operation.

{_POSIX_AIO_LISTIO_MAX}
The number of I/O operations that can be specified in a list I/O call.
Value: 2

{_POSIX_AIO_MAX}
The number of outstanding asynchronous I/O operations.
Value: 1

{_POSIX_ARG_MAX}
Maximum length of argument to the *exec* functions including environment data.
Value: 4 096

{_POSIX_CHILD_MAX}
Maximum number of simultaneous processes per real user ID.
Value: 25

{_POSIX_DELAYTIMER_MAX}
The number of timer expiration overruns.
Value: 32

{_POSIX_HOST_NAME_MAX}
Maximum length of a host name (not including the terminating null) as returned from the *gethostname*( ) function.
Value: 255

{_POSIX_LINK_MAX}
Maximum number of links to a single file.
Value: 8

{_POSIX_LOGIN_NAME_MAX}
The size of the storage required for a login name, in bytes (including the terminating null).
Value: 9

{_POSIX_MAX_CANON}
Maximum number of bytes in a terminal canonical input queue.
Value: 255

{_POSIX_MAX_INPUT}
Maximum number of bytes allowed in a terminal input queue.
Value: 255

MSG {_POSIX_MQ_OPEN_MAX}
The number of message queues that can be open for a single process.
Value: 8

MSG {_POSIX_MQ_PRIO_MAX}
The maximum number of message priorities supported by the implementation.
Value: 32

{_POSIX_NAME_MAX}
Maximum number of bytes in a filename (not including the terminating null of a filename string).
Value: 14

{_POSIX_NGROUPS_MAX}
Maximum number of simultaneous supplementary group IDs per process.
Value: 8

{_POSIX_OPEN_MAX}
A value one greater than the maximum value that the system may assign to a newly-created file descriptor.
Value: 20

{_POSIX_PATH_MAX}
Minimum number the implementation shall accept as the maximum number of bytes in a pathname.
Value: 256

{_POSIX_PIPE_BUF}
Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
Value: 512

{_POSIX_RE_DUP_MAX}
Maximum number of repeated occurrences of a BRE or ERE interval expression; see Section 9.3.6 (on page 185) and Section 9.4.6 (on page 189).
Value: 255

{_POSIX_RTSIG_MAX}
The number of realtime signal numbers reserved for application use.
Value: 8

{_POSIX_SEM_NSEMS_MAX}
The number of semaphores that a process may have.
Value: 256

{_POSIX_SEM_VALUE_MAX}
The maximum value a semaphore may have.
Value: 32 767

{_POSIX_SIGQUEUE_MAX}
The number of queued signals that a process may send and have pending at the receiver(s) at any time.
Value: 32

{_POSIX_SSIZE_MAX}
The value that can be stored in an object of type **ssize_t**.
Value: 32 767

SS | TSP {_POSIX_SS_REPL_MAX}
The number of replenishment operations that may be simultaneously pending for a particular sporadic server scheduler.
Value: 4

{_POSIX_STREAM_MAX}
The number of streams that one process can have open at one time.
Value: 8

{_POSIX_SYMLINK_MAX}
The number of bytes in a symbolic link.
Value: 255

{_POSIX_SYMLOOP_MAX}
The number of symbolic links that can be traversed in the resolution of a pathname in the absence of a loop.
Value: 8

{_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
The number of attempts made to destroy a thread's thread-specific data values on thread exit.
Value: 4

{_POSIX_THREAD_KEYS_MAX}
The number of data keys per process.
Value: 128

{_POSIX_THREAD_THREADS_MAX}
The number of threads per process.
Value: 64

{_POSIX_TIMER_MAX}
The per-process number of timers.
Value: 32

{_POSIX_TTY_NAME_MAX}
The size of the storage required for a terminal device name, in bytes (including the terminating null).
Value: 9

{_POSIX_TZNAME_MAX}
Maximum number of bytes supported for the name of a timezone (not of the *TZ* variable).
Value: 6

Note: The length given by {_POSIX_TZNAME_MAX} does not include the quoting characters mentioned in Section 8.3 (on page 174).

{_POSIX2_BC_BASE_MAX}
Maximum *obase* values allowed by the *bc* utility.
Value: 99

{_POSIX2_BC_DIM_MAX}
Maximum number of elements permitted in an array by the *bc* utility.
Value: 2 048

{_POSIX2_BC_SCALE_MAX}
Maximum *scale* value allowed by the *bc* utility.
Value: 99

{_POSIX2_BC_STRING_MAX}
Maximum length of a string constant accepted by the *bc* utility.
Value: 1 000

{_POSIX2_CHARCLASS_NAME_MAX}
Maximum number of bytes in a character class name.
Value: 14

{_POSIX2_COLL_WEIGHTS_MAX}
Maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order** keyword in the locale definition file; see Chapter 7 (on page 127).
Value: 2

{_POSIX2_EXPR_NEST_MAX}
Maximum number of expressions that can be nested within parentheses by the *expr* utility.
Value: 32

{_POSIX2_LINE_MAX}
Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <newline>.
Value: 2 048

{_POSIX2_RE_DUP_MAX}
Maximum number of repeated occurrences of a BRE or ERE interval expression; see Section 9.3.6 (on page 185) and Section 9.4.6 (on page 189).
Value: 255

XSI {_XOPEN_IOV_MAX}
Maximum number of **iovec** structures that one process has available for use with *readv*( ) or *writev*( ).
Value: 16

XSI {_XOPEN_NAME_MAX}
Maximum number of bytes in a filename (not including the terminating null of a filename string).
Value: 255

XSI {_XOPEN_PATH_MAX}
Minimum number the implementation shall accept as the maximum number of bytes in a pathname.
Value: 1 024

Numerical Limits

The **<limits.h>** header shall define the following macros and, except for {CHAR_BIT}, {LONG_BIT}, {MB_LEN_MAX}, and {WORD_BIT}, they shall be replaced by expressions that have the same type as would an expression that is an object of the corresponding type converted according to the integer promotions.

If an object of type **char** can hold negative values, the value of {CHAR_MIN} shall be the same as that of {SCHAR_MIN} and the value of {CHAR_MAX} shall be the same as that of {SCHAR_MAX}. Otherwise, the value of {CHAR_MIN} shall be 0 and the value of {CHAR_MAX} shall be the same as that of {UCHAR_MAX}.

{CHAR_BIT}
Number of bits in a type **char**.
CX Value: 8

{CHAR_MAX}
Maximum value for an object of type **char**.
Value: {UCHAR_MAX} or {SCHAR_MAX}

{CHAR_MIN}
Minimum value for an object of type **char**.
Value: {SCHAR_MIN} or 0

{INT_MAX}
Maximum value for an object of type **int**.
CX Minimum Acceptable Value: 2 147 483 647

{INT_MIN}
Minimum value for an object of type **int**.
CX Maximum Acceptable Value: –2 147 483 648

{LLONG_MAX}
Maximum value for an object of type **long long**.
Minimum Acceptable Value: +9 223 372 036 854 775 807

{LLONG_MIN}
Minimum value for an object of type **long long**.
CX Maximum Acceptable Value: –9 223 372 036 854 775 808

CX {LONG_BIT}
Number of bits in an object of type **long**.
Minimum Acceptable Value: 32

{LONG_MAX}
Maximum value for an object of type **long**.
Minimum Acceptable Value: +2 147 483 647

{LONG_MIN}
Minimum value for an object of type **long**.
CX Maximum Acceptable Value: –2 147 483 648

{MB_LEN_MAX}
Maximum number of bytes in a character, for any supported locale.
Minimum Acceptable Value: 1

{SCHAR_MAX}
Maximum value for an object of type **signed char**.
CX Value: +127

{SCHAR_MIN}
Minimum value for an object of type **signed char**.
CX Value: –128

{SHRT_MAX}
Maximum value for an object of type **short**.
Minimum Acceptable Value: +32 767

{SHRT_MIN}
Minimum value for an object of type **short**.
CX Maximum Acceptable Value: –32 768

CX {SSIZE_MAX}
Maximum value for an object of type **ssize_t**.
Minimum Acceptable Value: {_POSIX_SSIZE_MAX}

{UCHAR_MAX}
Maximum value for an object of type **unsigned char**.
CX Value: 255

{UINT_MAX}
Maximum value for an object of type **unsigned**.
CX Minimum Acceptable Value: 4 294 967 295

{ULLONG_MAX}
Maximum value for an object of type **unsigned long long**.
Minimum Acceptable Value: 18 446 744 073 709 551 615

{ULONG_MAX}
Maximum value for an object of type **unsigned long**.
Minimum Acceptable Value: 4 294 967 295

{USHRT_MAX}
Maximum value for an object of type **unsigned short**.
Minimum Acceptable Value: 65 535

CX {WORD_BIT}
Number of bits in an object of type **int**.
Minimum Acceptable Value: 32

Other Invariant Values

The **<limits.h>** header shall define the following symbolic constants:

{GETENTROPY_MAX}
The maximum value of the *length* argument in calls to the *getentropy*( ) function.
Minimum Acceptable Value: 256

{NL_ARGMAX}
Maximum value of *n* in conversion specifications using the "%*n*$" sequence in calls to the *printf*( ) and *scanf*( ) families of functions.
Minimum Acceptable Value: 9

XSI {NL_LANGMAX}
Maximum number of bytes in a *LANG* name.
Minimum Acceptable Value: 14

{NL_MSGMAX}
Maximum message number.
Minimum Acceptable Value: 32 767

{NL_SETMAX}
Maximum set number.
Minimum Acceptable Value: 255

{NL_TEXTMAX}
Maximum number of bytes in a message string.
Minimum Acceptable Value: {_POSIX2_LINE_MAX}

{NSIG_MAX}
Maximum possible return value of *sysconf*(_SC_NSIG). See XSH *sysconf*( ). The value of {NSIG_MAX} shall be no greater than the number of signals that the **sigset_t** type (see **<signal.h>**) is capable of representing, ignoring any restrictions imposed by *sigfillset*( ) or *sigaddset*( ).

XSI {NZERO}
Default process priority.
Minimum Acceptable Value: 20

APPLICATION USAGE

None.

RATIONALE

A request was made to reduce the value of {_POSIX_LINK_MAX} from the value of 8 specified for it in the POSIX.1-1990 standard to 2. The standard developers decided to deny this request for several reasons:

- They wanted to avoid making any changes to the standard that could break conforming applications, and the requested change could have that effect.
- The use of multiple hard links to a file cannot always be replaced with use of symbolic links. Symbolic links are semantically different from hard links in that they associate a pathname with another pathname rather than a pathname with a file. This has implications for access control, file permanence, and transparency.

- The original standard developers had considered the issue of allowing for implementations that did not in general support hard links, and decided that this would reduce consensus on the standard.

Systems that support historical versions of the development option of the ISO POSIX-2 standard retain the name {_POSIX2_RE_DUP_MAX} as an alias for {_POSIX_RE_DUP_MAX}.

{NSIG_MAX}
Some historical implementations provided compile-time constants NSIG or SIGMAX to define the maximum number of signals the implementation supported, but these values did not necessarily reflect the number of signals that could be handled using a **sigset_t**. With the addition of real-time signals and the desire by some applications to be able to allocate additional real-time signals at run-time, neither of these constants provided a useable, portable value. {NSIG_MAX} was added to the standard to allow applications to determine the maximum number of signals that an implementation will support based on the size of the **sigset_t** type (defined in **<signal.h>**).

{PATH_MAX}
IEEE PASC Interpretation 1003.1 #15 addressed the inconsistency in the standard with the definition of pathname and the description of {PATH_MAX}, allowing application developers to allocate either {PATH_MAX} or {PATH_MAX}+1 bytes. The inconsistency has been removed by correction to the {PATH_MAX} definition to include the null character. With this change, applications that previously allocated {PATH_MAX} bytes will continue to succeed.

{SYMLINK_MAX}
This symbol refers to space for data that is stored in the file system, as opposed to {PATH_MAX} which is the length of a name that can be passed to a function. In some existing implementations, the pathnames pointed to by symbolic links are stored in the *inode*s of the links, so it is important that {SYMLINK_MAX} not be constrained to be as large as {PATH_MAX}.

The maximum values for {SCHAR_MIN}, {SHRT_MIN}, {LONG_MIN} and {LLONG_MIN} differ from the ISO C standard because POSIX.1 requires two's complement representation for the corresponding integer types. The maximum value for {INT_MIN} differs both for that reason and because POSIX.1 requires that **int** has a width of at least 32 bits. See also the RATIONALE section for **<stdint.h>**.

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 7 (on page 127), **<stdint.h>**, **<stdio.h>**, **<unistd.h>**

XSH Section 2.2 (on page 496), *fpathconf*( ), *getrlimit*( ), *sysconf*( )

CHANGE HISTORY

First released in Issue 1.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

{FILESIZEBITS} is added for the Large File Summit extensions.

The minimum acceptable values for {INT_MAX}, {INT_MIN}, and {UINT_MAX} are changed to make 32-bit values the minimum requirement.

The entry is restructured to improve readability.

Issue 6

The Open Group Corrigendum U033/4 is applied. The wording is made clear for {CHAR_MIN}, {INT_MIN}, {LONG_MIN}, {SCHAR_MIN}, and {SHRT_MIN} that these are maximum acceptable values.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The minimum value for {CHILD_MAX} is 25. This is a FIPS requirement.
- The minimum value for {OPEN_MAX} is 20. This is a FIPS requirement.
- The minimum value for {NGROUPS_MAX} is 8. This is also a FIPS requirement.

Symbolic constants are added for {_POSIX_SYMLINK_MAX}, {_POSIX_SYMLOOP_MAX}, {_POSIX_RE_DUP_MAX}, {RE_DUP_MAX}, {SYMLOOP_MAX}, and {SYMLINK_MAX}.

The following values are added for alignment with IEEE Std 1003.1d-1999:

{_POSIX_SS_REPL_MAX}
{SS_REPL_MAX}
{POSIX_ALLOC_SIZE_MIN}
{POSIX_REC_INCR_XFER_SIZE}
{POSIX_REC_MAX_XFER_SIZE}
{POSIX_REC_MIN_XFER_SIZE}
{POSIX_REC_XFER_ALIGN}

Reference to CLOCK_MONOTONIC is added in the description of {_POSIX_CLOCKRES_MIN} for alignment with IEEE Std 1003.1j-2000.

The constants {LLONG_MIN}, {LLONG_MAX}, and {ULLONG_MAX} are added for alignment with the ISO/IEC 9899: 1999 standard.

The following values are added for alignment with IEEE Std 1003.1q-2000:

{_POSIX_TRACE_EVENT_NAME_MAX}
{_POSIX_TRACE_NAME_MAX}
{_POSIX_TRACE_SYS_MAX}
{_POSIX_TRACE_USER_EVENT_MAX}
{TRACE_EVENT_NAME_MAX}
{TRACE_NAME_MAX}
{TRACE_SYS_MAX}
{TRACE_USER_EVENT_MAX}

The new limits {_XOPEN_NAME_MAX} and {_XOPEN_PATH_MAX} are added as minimum values for {PATH_MAX} and {NAME_MAX} limits on XSI-conformant systems.

The LEGACY symbols {PASS_MAX} and {TMP_MAX} are removed.

The values for the limits {CHAR_BIT}, {SCHAR_MAX}, and {UCHAR_MAX} are now required to be 8, +127, and 255, respectively.

The value for the limit {CHAR_MAX} is now {UCHAR_MAX} or {SCHAR_MAX}.

The value for the limit {CHAR_MIN} is now {SCHAR_MIN} or zero.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/10 is applied, correcting the value of {_POSIX_CHILD_MAX} from 6 to 25. This is for FIPS 151-2 alignment.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/19 is applied, updating the values for {INT_MAX}, {UINT_MAX}, and {INT_MIN} to be CX extensions over the ISO C standard, and correcting {WORD_BIT} from 16 to 32.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/20 is applied, removing {CHARCLASS_NAME_MAX} from the ``Other Invariant Values'' section (it also occurs under ``Runtime Increasable Values'').

Issue 7

Austin Group Interpretations 1003.1-2001 #143 and #160 are applied.

Austin Group Interpretation 1003.1-2001 #173 is applied, updating the descriptions of {TRACE_EVENT_NAME_MAX} and {TRACE_NAME_MAX} to not include the terminating null.

SD5-XBD-ERN-36 is applied, changing the description of {RE_DUP_MAX}.

SD5-XBD-ERN-90 is applied.

{NL_NMAX} is removed; it should have been removed in Issue 6.

The Trace option values are marked obsolescent.

The {ATEXIT_MAX}, {LONG_BIT}, {NL_MSGMAX}, {NL_SETMAX}, {NL_TEXTMAX}, and {WORD_BIT} values are moved from the XSI option to the Base.

The AIO_* and _POSIX_AIO_* values are moved from the Asynchronous Input and Output option to the Base.

The {_POSIX_RTSIG_MAX}, {_POSIX_SIGQUEUE_MAX}, {RTSIG_MAX}, and {SIGQUEUE_MAX} values are moved from the Realtime Signals Extension option to the Base.

Functionality relating to the Threads and Timers options is moved to the Base.

This reference page is clarified with respect to macros and symbolic constants.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0052 [108], XBD/TC1-2008/0053 [291], XBD/TC1-2008/0054 [182,427], XBD/TC1-2008/0055 [291], XBD/TC1-2008/0056 [371], XBD/TC1-2008/0057 [291], XBD/TC1-2008/0058 [108], and XBD/TC1-2008/0059 [291] are applied.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0061 [666] is applied.

Issue 8

Austin Group Defect 741 is applied, adding {NSIG_MAX}.

Austin Group Defect 1108 is applied, changing the maximum allowed value for all signed integer minimum limits.

Austin Group Defect 1122 is applied, adding {TEXTDOMAIN_MAX}.

Austin Group Defect 1134 is applied, adding {GETENTROPY_MAX}.

Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899: 2018 standard.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.

Austin Group Defect 1446 is applied, changing the introductory paragraphs of the DESCRIPTION to include mention of *setrlimit*( ) and *getrlimit*( ).

NAME

locale.h — category macros

SYNOPSIS

```
#include <locale.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to enable the visibility of these symbols in this header.

The **<locale.h>** header shall define the **lconv** structure, which shall include at least the following members. (See the definitions of *LC_MONETARY* in Section 7.3.3 (on page 147) and Section 7.3.4 (on page 151).)

```
char    *currency_symbol
char    *decimal_point
char     frac_digits
char    *grouping
char    *int_curr_symbol
char     int_frac_digits
char     int_n_cs_precedes
char     int_n_sep_by_space
char     int_n_sign_posn
char     int_p_cs_precedes
char     int_p_sep_by_space
char     int_p_sign_posn
char    *mon_decimal_point
char    *mon_grouping
char    *mon_thousands_sep
char    *negative_sign
char     n_cs_precedes
char     n_sep_by_space
char     n_sign_posn
char    *positive_sign
char     p_cs_precedes
char     p_sep_by_space
char     p_sign_posn
char    *thousands_sep
```

The **<locale.h>** header shall define NULL (as described in **<stddef.h>**) and at least the following as macros:

LC_ALL
LC_COLLATE
LC_CTYPE
CX LC_MESSAGES
LC_MONETARY
LC_NUMERIC
LC_TIME

which shall expand to integer constant expressions with distinct values for use as the first argument to the *setlocale*( ) function.

Additional macro definitions, beginning with the characters *LC_* and an uppercase letter, may also be specified by the implementation.

CX The **<locale.h>** header shall contain at least the following macros representing bitmasks for use with the *newlocale*( ) function for each supported locale category:

LC_COLLATE_MASK
LC_CTYPE_MASK
LC_MESSAGES_MASK
LC_MONETARY_MASK
LC_NUMERIC_MASK
LC_TIME_MASK

In addition, a macro to set the bits for all categories set shall be defined:

LC_ALL_MASK

The **<locale.h>** header shall define LC_GLOBAL_LOCALE, a special locale object descriptor used by the *duplocale*( ) and *uselocale*( ) functions.

The **<locale.h>** header shall define the **locale_t** type, representing a locale object.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

```
CX  locale_t      duplocale(locale_t);
    void          freelocale(locale_t);
    const char   *getlocalename_l(int, locale_t);
    struct lconv *localeconv(void);
CX  locale_t      newlocale(int, const char *, locale_t);
    char         *setlocale(int, const char *);
CX  locale_t      uselocale (locale_t);
```

APPLICATION USAGE

None.

RATIONALE

It is suggested that each category macro name for use in *setlocale*( ) have a corresponding macro name ending in *_MASK* for use in *newlocale*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 8 (on page 167), **<stddef.h>**

XSH *duplocale*( ), *freelocale*( ), *getlocalename_l*( ), *localeconv*( ), *newlocale*( ), *setlocale*( ), *uselocale*( )

CHANGE HISTORY

First released in Issue 3.

Included for alignment with the ISO C standard.

Issue 6

The **lconv** structure is expanded with new members (**int_n_cs_precedes**, **int_n_sep_by_space**, **int_n_sign_posn**, **int_p_cs_precedes**, **int_p_sep_by_space**, and **int_p_sign_posn**) for alignment with the ISO/IEC 9899: 1999 standard.

Extensions beyond the ISO C standard are marked.

Issue 7

The *duplocale*(), *freelocale*(), *newlocale*(), and *uselocale*() functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

This reference page is clarified with respect to macros and symbolic constants.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0060 [301,427] is applied.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0062 [781] is applied.

Issue 8

Austin Group Defect 1220 is applied, adding *getlocalename_l*().

NAME

math.h — mathematical declarations

SYNOPSIS

```
#include <math.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to enable the visibility of these symbols in this header.

The **<math.h>** header shall define at least the following types:

float_t A real-floating type at least as wide as **float**.

double_t A real-floating type at least as wide as **double**, and at least as wide as **float_t**.

If FLT_EVAL_METHOD equals 0, **float_t** and **double_t** shall be **float** and **double**, respectively; if FLT_EVAL_METHOD equals 1, they shall both be **double**; if FLT_EVAL_METHOD equals 2, they shall both be **long double**; for other values of FLT_EVAL_METHOD, they are otherwise implementation-defined.

The **<math.h>** header shall define the following macros, where real-floating indicates that the argument shall be an expression of real-floating type:

```
int fpclassify(real-floating x);
int isfinite(real-floating x);
int isgreater(real-floating x, real-floating y);
int isgreaterequal(real-floating x, real-floating y);
int isinf(real-floating x);
int isless(real-floating x, real-floating y);
int islessequal(real-floating x, real-floating y);
int islessgreater(real-floating x, real-floating y);
int isnan(real-floating x);
int isnormal(real-floating x);
int isunordered(real-floating x, real-floating y);
int signbit(real-floating x);
```

XSI The **<math.h>** header shall define the following symbolic constants. Where the constant name ends in `'l'` (lower case ell), the values shall have type **long double**; otherwise the values shall have type **double**. The values shall be accurate to the precision of their type. If the implementation supports FLT_EVAL_METHOD values other than 0 or 1, the values shall either include an explicit cast for that type, or be expressed as hexadecimal floating constants.

Double	Long Double	Value
M_E	M_El	Value of e
M_EGAMMA	M_EGAMMAl	Value of γ, Euler-Mascheroni constant
M_LOG2E	M_LOG2El	Value of $\log_2 e$
M_LOG10E	M_LOG10El	Value of $\log_{10} e$
M_LN2	M_LN2l	Value of $\log_e 2$
M_LN10	M_LN10l	Value of $\log_e 10$
M_PHI	M_PHIl	Value of ϕ, $(1+\sqrt{5})/2$, golden ratio constant
M_PI	M_PIl	Value of π
M_PI_2	M_PI_2l	Value of $\pi/2$
M_PI_4	M_PI_4l	Value of $\pi/4$
M_1_PI	M_1_PIl	Value of $1/\pi$
M_1_SQRTPI	M_1_SQRTPIl	Value of $1/\sqrt{\pi}$
M_2_PI	M_2_PIl	Value of $2/\pi$
M_2_SQRTPI	M_2_SQRTPIl	Value of $2/\sqrt{\pi}$
M_SQRT2	M_SQRT2l	Value of $\sqrt{2}$
M_SQRT3	M_SQRT3l	Value of $\sqrt{3}$
M_SQRT1_2	M_SQRT1_2l	Value of $1/\sqrt{2}$
M_SQRT1_3	M_SQRT1_3l	Value of $1/\sqrt{3}$

The **<math.h>** header shall define the following macros:

HUGE_VAL A positive **double** constant expression, not necessarily representable as a **float**. Used as an error value returned by the mathematics library. HUGE_VAL evaluates to +infinity on systems supporting IEEE Std 754-1985.

HUGE_VALF A positive **float** constant expression. Used as an error value returned by the mathematics library. HUGE_VALF evaluates to +infinity on systems supporting IEEE Std 754-1985.

HUGE_VALL A positive **long double** constant expression. Used as an error value returned by the mathematics library. HUGE_VALL evaluates to +infinity on systems supporting IEEE Std 754-1985.

INFINITY A constant expression of type **float** representing positive or unsigned infinity, if available; else a positive constant of type **float** that overflows at translation time.

NAN A constant expression of type **float** representing a quiet NaN. This macro is only defined if the implementation supports quiet NaNs for the **float** type.

The following macros shall be defined for number classification. They represent the mutually-exclusive kinds of floating-point values. They expand to integer constant expressions with distinct values. Additional implementation-defined floating-point classifications, with macro definitions beginning with FP_ and an uppercase letter, may also be specified by the implementation.

FP_INFINITE
FP_NAN
FP_NORMAL
FP_SUBNORMAL
FP_ZERO

The following optional macros indicate whether the *fma*( ) family of functions are fast compared

with direct code:

FP_FAST_FMA
FP_FAST_FMAF
FP_FAST_FMAL

If defined, the FP_FAST_FMA macro shall expand to the integer constant 1 and shall indicate that the *fma*( ) function generally executes about as fast as, or faster than, a multiply and an add of **double** operands. If undefined, the speed of execution is unspecified. The other macros have the equivalent meaning for the **float** and **long double** versions.

The following macros shall expand to integer constant expressions whose values are returned by *ilogb*(*x*) if *x* is zero or NaN, respectively. The value of FP_ILOGB0 shall be either {INT_MIN} or −{INT_MAX}. The value of FP_ILOGBNAN shall be either {INT_MAX} or {INT_MIN}.

FP_ILOGB0
FP_ILOGBNAN

The following macros shall expand to the integer constants 1 and 2, respectively;

MATH_ERRNO
MATH_ERREXCEPT

The following macro shall expand to an expression that has type **int** and the value MATH_ERRNO, MATH_ERREXCEPT, or the bitwise-inclusive OR of both:

math_errhandling

The value of math_errhandling is constant for the duration of the program. It is unspecified whether math_errhandling is a macro or an identifier with external linkage. If a macro definition is suppressed or a program defines an identifier with the name math_errhandling , the behavior is undefined. If the expression (math_errhandling & MATH_ERREXCEPT) can be non-zero, the implementation shall define the macros FE_DIVBYZERO, FE_INVALID, and FE_OVERFLOW in **<fenv.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
double      acos(double);
float       acosf(float);
double      acosh(double);
float       acoshf(float);
long double acoshl(long double);
long double acosl(long double);
double      asin(double);
float       asinf(float);
double      asinh(double);
float       asinhf(float);
long double asinhl(long double);
long double asinl(long double);
double      atan(double);
double      atan2(double, double);
float       atan2f(float, float);
long double atan2l(long double, long double);
float       atanf(float);
```



```
10610         double      atanh(double);
10611         float       atanhf(float);
10612         long double atanhl(long double);
10613         long double atanl(long double);
10614         double      cbrt(double);
10615         float       cbrtf(float);
10616         long double cbrtl(long double);
10617         double      ceil(double);
10618         float       ceilf(float);
10619         long double ceill(long double);
10620         double      copysign(double, double);
10621         float       copysignf(float, float);
10622         long double copysignl(long double, long double);
10623         double      cos(double);
10624         float       cosf(float);
10625         double      cosh(double);
10626         float       coshf(float);
10627         long double coshl(long double);
10628         long double cosl(long double);
10629         double      erf(double);
10630         double      erfc(double);
10631         float       erfcf(float);
10632         long double erfcl(long double);
10633         float       erff(float);
10634         long double erfl(long double);
10635         double      exp(double);
10636         double      exp2(double);
10637         float       exp2f(float);
10638         long double exp2l(long double);
10639         float       expf(float);
10640         long double expl(long double);
10641         double      expm1(double);
10642         float       expm1f(float);
10643         long double expm1l(long double);
10644         double      fabs(double);
10645         float       fabsf(float);
10646         long double fabsl(long double);
10647         double      fdim(double, double);
10648         float       fdimf(float, float);
10649         long double fdiml(long double, long double);
10650         double      floor(double);
10651         float       floorf(float);
10652         long double floorl(long double);
10653         double      fma(double, double, double);
10654         float       fmaf(float, float, float);
10655         long double fmal(long double, long double, long double);
10656         double      fmax(double, double);
10657         float       fmaxf(float, float);
10658         long double fmaxl(long double, long double);
10659         double      fmin(double, double);
10660         float       fminf(float, float);
10661         long double fminl(long double, long double);
```



```
10662          double      fmod(double, double);
10663          float       fmodf(float, float);
10664          long double fmodl(long double, long double);
10665          double      frexp(double, int *);
10666          float       frexpf(float, int *);
10667          long double frexpl(long double, int *);
10668          double      hypot(double, double);
10669          float       hypotf(float, float);
10670          long double hypotl(long double, long double);
10671          int         ilogb(double);
10672          int         ilogbf(float);
10673          int         ilogbl(long double);
10674    XSI   double      j0(double);
10675          double      j1(double);
10676          double      jn(int, double);
10677          double      ldexp(double, int);
10678          float       ldexpf(float, int);
10679          long double ldexpl(long double, int);
10680          double      lgamma(double);
10681          float       lgammaf(float);
10682          long double lgammal(long double);
10683          long long   llrint(double);
10684          long long   llrintf(float);
10685          long long   llrintl(long double);
10686          long long   llround(double);
10687          long long   llroundf(float);
10688          long long   llroundl(long double);
10689          double      log(double);
10690          double      log10(double);
10691          float       log10f(float);
10692          long double log10l(long double);
10693          double      log1p(double);
10694          float       log1pf(float);
10695          long double log1pl(long double);
10696          double      log2(double);
10697          float       log2f(float);
10698          long double log2l(long double);
10699          double      logb(double);
10700          float       logbf(float);
10701          long double logbl(long double);
10702          float       logf(float);
10703          long double logl(long double);
10704          long        lrint(double);
10705          long        lrintf(float);
10706          long        lrintl(long double);
10707          long        lround(double);
10708          long        lroundf(float);
10709          long        lroundl(long double);
10710          double      modf(double, double *);
10711          float       modff(float, float *);
10712          long double modfl(long double, long double *);
10713          double      nan(const char *);
```



```
10714       float       nanf(const char *);
10715       long double nanl(const char *);
10716       double      nearbyint(double);
10717       float       nearbyintf(float);
10718       long double nearbyintl(long double);
10719       double      nextafter(double, double);
10720       float       nextafterf(float, float);
10721       long double nextafterl(long double, long double);
10722       double      nexttoward(double, long double);
10723       float       nexttowardf(float, long double);
10724       long double nexttowardl(long double, long double);
10725       double      pow(double, double);
10726       float       powf(float, float);
10727       long double powl(long double, long double);
10728       double      remainder(double, double);
10729       float       remainderf(float, float);
10730       long double remainderl(long double, long double);
10731       double      remquo(double, double, int *);
10732       float       remquof(float, float, int *);
10733       long double remquol(long double, long double, int *);
10734       double      rint(double);
10735       float       rintf(float);
10736       long double rintl(long double);
10737       double      round(double);
10738       float       roundf(float);
10739       long double roundl(long double);
10740       double      scalbln(double, long);
10741       float       scalblnf(float, long);
10742       long double scalblnl(long double, long);
10743       double      scalbn(double, int);
10744       float       scalbnf(float, int);
10745       long double scalbnl(long double, int);
10746       double      sin(double);
10747       float       sinf(float);
10748       double      sinh(double);
10749       float       sinhf(float);
10750       long double sinhl(long double);
10751       long double sinl(long double);
10752       double      sqrt(double);
10753       float       sqrtf(float);
10754       long double sqrtl(long double);
10755       double      tan(double);
10756       float       tanf(float);
10757       double      tanh(double);
10758       float       tanhf(float);
10759       long double tanhl(long double);
10760       long double tanl(long double);
10761       double      tgamma(double);
10762       float       tgammaf(float);
10763       long double tgammal(long double);
10764       double      trunc(double);
10765       float       truncf(float);
```



```
10766          long double truncl(long double);
10767 XSI      double      y0(double);
10768          double      y1(double);
10769          double      yn(int, double);
```

10770 The following external variable shall be defined:

```
10771 XSI      extern int signgam;
```

10772 The behavior of each of the functions defined in **<math.h>** is specified in the System Interfaces
10773 volume of POSIX.1-2024 for all representable values of its input arguments, except where stated
10774 otherwise. Each function shall execute as if it were a single operation without generating any
10775 externally visible exceptional conditions.

10776 **APPLICATION USAGE**

10777 The FP_CONTRACT pragma can be used to allow (if the state is on) or disallow (if the state is
10778 off) the implementation to contract expressions. Each pragma can occur either outside external
10779 declarations or preceding all explicit declarations and statements inside a compound statement.
10780 When outside external declarations, the pragma takes effect from its occurrence until another
10781 FP_CONTRACT pragma is encountered, or until the end of the translation unit. When inside a
10782 compound statement, the pragma takes effect from its occurrence until another FP_CONTRACT
10783 pragma is encountered (including within a nested compound statement), or until the end of the
10784 compound statement; at the end of a compound statement the state for the pragma is restored to
10785 its condition just before the compound statement. If this pragma is used in any other context, the
10786 behavior is undefined. The default state (on or off) for the pragma is implementation-defined.

10787 **RATIONALE**

10788 Before the ISO/IEC 9899:1999 standard, the math library was defined only for the floating type
10789 **double**. All the names formed by appending `'f'` or `'l'` to a name in **<math.h>** were reserved
10790 to allow for the definition of **float** and **long double** libraries; and the ISO/IEC 9899:1999
10791 standard provided for all three versions of math functions.

10792 The functions *ecvt*(), *fcvt*(), and *gcvt*() have been dropped from the ISO C standard since their
10793 capability is available through *sprintf*().

10794 The requirement for an explicit cast or representation via hexadecimal floating constants is to
10795 guarantee that even when FLT_EVAL_METHOD is neither 0 nor 1, the expression
10796 `(double)M_PI == M_PI` will always hold true. Earlier versions of this standard did not make
10797 this requirement, because they lacked the `'l'` (lower case ell) versions of the constants and were
10798 allowed to provide M_PI with long double precision depending on FLT_EVAL_METHOD.

10799 **FUTURE DIRECTIONS**

10800 None.

10801 **SEE ALSO**

10802 **<float.h>**, **<stddef.h>**, **<sys/types.h>**

10803 XSH Section 2.2 (on page 496), *acos*(), *acosh*(), *asin*(), *asinh*(), *atan*(), *atan2*(), *atanh*(), *cbrt*(),
10804 *ceil*(), *copysign*(), *cos*(), *cosh*(), *erf*(), *erfc*(), *exp*(), *exp2*(), *expm1*(), *fabs*(), *fdim*(), *floor*(), *fma*(),
10805 *fmax*(), *fmin*(), *fmod*(), *fpclassify*(), *frexp*(), *hypot*(), *ilogb*(), *isfinite*(), *isgreater*(), *isinf*(), *isnan*(),
10806 *isnormal*(), *isunordered*(), *j0*(), *ldexp*(), *lgamma*(), *llrint*(), *llround*(), *log*(), *log10*(), *log1p*(), *log2*(),
10807 *logb*(), *lrint*(), *lround*(), *modf*(), *nan*(), *nearbyint*(), *nextafter*(), *pow*(), *remainder*(), *remquo*(),
10808 *rint*(), *round*(), *scalbln*(), *signbit*(), *sin*(), *sinh*(), *sqrt*(), *tan*(), *tanh*(), *tgamma*(), *trunc*(), *y0*()

10809 **CHANGE HISTORY**

10810 First released in Issue 1.

10811 **Issue 6**

10812 This reference page is updated to align with the ISO/IEC 9899: 1999 standard.

10813 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/21 is applied, making it clear that the
10814 meaning of the FP_FAST_FMA macro is unspecified if the macro is undefined.

10815 **Issue 7**

10816 ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #47 (SD5-XBD-ERN-52) is applied,
10817 clarifying the wording of the FP_FAST_FMA macro.

10818 The MAXFLOAT constant is marked obsolescent.

10819 This reference page is clarified with respect to macros and symbolic constants.

10820 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0063 [801] and XBD/TC2-2008/0064
10821 [801] are applied.

10822 **Issue 8**

10823 Austin Group Defect 828 is applied, adding **long double** counterparts with names ending in
10824 `'l'` to the **double** constants whose names begin with `"M_"`, and changing the representation
10825 requirements for those constants.

10826 Austin Group Defect 1302 is applied, changing ``provides'' to ``provided''.

10827 Austin Group Defect 1330 is applied, removing the obsolescent MAXFLOAT.

10828 Austin Group Defect 1503 is applied, adding M_1_SQRTPI, M_EGAMMA, M_PHI, M_SQRT1_3,
10829 and M_SQRT3.

NAME

monetary.h — monetary types

SYNOPSIS

```
#include <monetary.h>
```

DESCRIPTION

The **<monetary.h>** header shall define the **locale_t** type as described in **<locale.h>**.

The **<monetary.h>** header shall define the **size_t** type as described in **<stddef.h>**.

The **<monetary.h>** header shall define the **ssize_t** type as described in **<sys/types.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

```
ssize_t  strfmon(char *restrict, size_t, const char *restrict, ...);
ssize_t  strfmon_l(char *restrict, size_t, locale_t,
             const char *restrict, ...);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<locale.h>, **<stddef.h>**, **<sys/types.h>**

XSH *strfmon*( )

CHANGE HISTORY

First released in Issue 4.

Issue 6

The **restrict** keyword is added to the prototype for *strfmon*( ).

Issue 7

The **<monetary.h>** header is moved from the XSI option to the Base.

The *strfmon_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

A declaration for the **locale_t** type is added.

NAME

mqueue.h — message queues (**REALTIME**)

SYNOPSIS

MSG

```
#include <mqueue.h>
```

DESCRIPTION

The **<mqueue.h>** header shall define the **mqd_t** type, which is used for message queue descriptors. This is not an array type.

The **<mqueue.h>** header shall define the **size_t** and **ssize_t** types as described in **<sys/types.h>**.

The **<mqueue.h>** header shall define the **struct timespec** structure as described in **<time.h>**.

The tag **sigevent** shall be declared as naming an incomplete structure type, the contents of which are described in the **<signal.h>** header.

The **<mqueue.h>** header shall define the **mq_attr** structure, which is used in getting and setting the attributes of a message queue. Attributes are initially set when the message queue is created. An **mq_attr** structure shall have at least the following fields:

`long`	`mq_flags`	Message queue flags.
`long`	`mq_maxmsg`	Maximum number of messages.
`long`	`mq_msgsize`	Maximum message size.
`long`	`mq_curmsgs`	Number of messages currently queued.

The **<mqueue.h>** header shall define O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_EXCL, and O_NONBLOCK as described in **<fcntl.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int      mq_close(mqd_t);
int      mq_getattr(mqd_t, struct mq_attr *);
int      mq_notify(mqd_t, const struct sigevent *);
mqd_t    mq_open(const char *, int, ...);
ssize_t  mq_receive(mqd_t, char *, size_t, unsigned *);
int      mq_send(mqd_t, const char *, size_t, unsigned);
int      mq_setattr(mqd_t, const struct mq_attr *restrict,
             struct mq_attr *restrict);
ssize_t  mq_timedreceive(mqd_t, char *restrict, size_t,
             unsigned *restrict, const struct timespec *restrict);
int      mq_timedsend(mqd_t, const char *, size_t, unsigned,
             const struct timespec *);
int      mq_unlink(const char *);
```

Inclusion of the **<mqueue.h>** header may make visible symbols defined in the headers **<fcntl.h>**, **<signal.h>**, and **<time.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<fcntl.h>, **<signal.h>**, **<sys/types.h>**, **<time.h>**

XSH *mq_close*( ), *mq_getattr*( ), *mq_notify*( ), *mq_open*( ), *mq_receive*( ), *mq_send*( ), *mq_setattr*( ), *mq_unlink*( )

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The **<mqueue.h>** header is marked as part of the Message Passing option.

The *mq_timedreceive*( ) and *mq_timedsend*( ) functions are added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the prototypes for *mq_setattr*( ) and *mq_timedreceive*( ).

Issue 7

Type and structure declarations are added.

Issue 8

Austin Group Defect 593 is applied, adding O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_EXCL, and O_NONBLOCK.

Austin Group Defect 1282 is applied, removing the requirement for **<mqueue.h>** to define **pthread_attr_t**.

NAME

ndbm.h — definitions for ndbm database operations

SYNOPSIS

XSI

```
#include <ndbm.h>
```

DESCRIPTION

The **<ndbm.h>** header shall define the **datum** type as a structure, which shall include at least the following members:

`void`	`*dptr`	A pointer to the application's data.
`size_t`	`dsize`	The size of the object pointed to by *dptr*.

The **<ndbm.h>** header shall define the **size_t** type as described in **<stddef.h>**.

The **<ndbm.h>** header shall define the **DBM** type.

The **<ndbm.h>** header shall define the following symbolic constants as possible values for the *store_mode* argument to *dbm_store*( ):

DBM_INSERT	Insertion of new entries only.
DBM_REPLACE	Allow replacing existing entries.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int     dbm_clearerr(DBM *);
void    dbm_close(DBM *);
int     dbm_delete(DBM *, datum);
int     dbm_error(DBM *);
datum   dbm_fetch(DBM *, datum);
datum   dbm_firstkey(DBM *);
datum   dbm_nextkey(DBM *);
DBM    *dbm_open(const char *, int, mode_t);
int     dbm_store(DBM *, datum, datum, int);
```

The **<ndbm.h>** header shall define the **mode_t** type through **typedef**, as described in **<sys/types.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<stddef.h>, **<sys/types.h>**

XSH *dbm_clearerr*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

References to the definitions of **size_t** and **mode_t** are added to the DESCRIPTION.

Issue 7

This reference page is clarified with respect to macros and symbolic constants.

NAME

net/if.h — sockets local interfaces

SYNOPSIS

```
#include <net/if.h>
```

DESCRIPTION

The **<net/if.h>** header shall define the **if_nameindex** structure, which shall include at least the following members:

```
unsigned  if_index   Numeric index of the interface.
char      *if_name   Null-terminated name of the interface.
```

The **<net/if.h>** header shall define the following symbolic constant for the length of a buffer containing an interface name (including the terminating NULL character):

IF_NAMESIZE Interface name length.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
void                 if_freenameindex(struct if_nameindex *);
char                *if_indextoname(unsigned, char *);
struct if_nameindex *if_nameindex(void);
unsigned             if_nametoindex(const char *);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH *if_freenameindex*( ), *if_indextoname*( ), *if_nameindex*( ), *if_nametoindex*( )

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7

This reference page is clarified with respect to macros and symbolic constants.

10995 **NAME**

10996 netdb.h — definitions for network database operations

10997 **SYNOPSIS**

10998 `#include <netdb.h>`

10999 **DESCRIPTION**

11000 The **<netdb.h>** header shall define the **hostent** structure, which shall include at least the
11001 following members:

11002 `char *h_name`	Official name of the host.
11003 `char **h_aliases`	A pointer to an array of pointers to 11004 alternative host names, terminated by a 11005 null pointer.
11006 `int h_addrtype`	Address type.
11007 `int h_length`	The length, in bytes, of the address.
11008 `char **h_addr_list`	A pointer to an array of pointers to network 11009 addresses (in network byte order) for the host, 11010 terminated by a null pointer.

11011 The **<netdb.h>** header shall define the **netent** structure, which shall include at least the
11012 following members:

11013 `char *n_name`	Official, fully-qualified (including the 11014 domain) name of the host.
11015 `char **n_aliases`	A pointer to an array of pointers to 11016 alternative network names, terminated by a 11017 null pointer.
11018 `int n_addrtype`	The address type of the network.
11019 `uint32_t n_net`	The network number, in host byte order.

11020 The **<netdb.h>** header shall define the **uint32_t** type as described in **<inttypes.h>**.

11021 The **<netdb.h>** header shall define the **protoent** structure, which shall include at least the
11022 following members:

11023 `char *p_name`	Official name of the protocol.
11024 `char **p_aliases`	A pointer to an array of pointers to 11025 alternative protocol names, terminated by 11026 a null pointer.
11027 `int p_proto`	The protocol number.

11028 The **<netdb.h>** header shall define the **servent** structure, which shall include at least the
11029 following members:

11030 `char *s_name`	Official name of the service.
11031 `char **s_aliases`	A pointer to an array of pointers to 11032 alternative service names, terminated by 11033 a null pointer.
11034 `int s_port`	A value which, when converted to uint16_t, 11035 yields the port number in network byte order 11036 at which the service resides.
11037 `char *s_proto`	The name of the protocol to use when 11038 contacting the service.

11039 The **<netdb.h>** header shall define the IPPORT_RESERVED symbolic constant with the value of
11040 the highest reserved Internet port number.

11041 **Address Information Structure**

11042 The **<netdb.h>** header shall define the **addrinfo** structure, which shall include at least the
11043 following members:

11044 `int`	`ai_flags`	Input flags.
11045 `int`	`ai_family`	Address family of socket.
11046 `int`	`ai_socktype`	Socket type.
11047 `int`	`ai_protocol`	Protocol of socket.
11048 `socklen_t`	`ai_addrlen`	Length of socket address.
11049 `struct sockaddr`	`*ai_addr`	Socket address of socket.
11050 `char`	`*ai_canonname`	Canonical name of service location.
11051 `struct addrinfo`	`*ai_next`	Pointer to next in list.

11052 The **addrinfo** structure shall not include any additional members which have a floating-point
11053 type if an object of that type with an all-bits-zero representation does not have the value 0.0.
11054 MX Implementations that define __STDC_IEC_559__ are required to treat the all-zero bit pattern for
11055 a floating point object as a representation of 0.0, and may therefore have floating-point type
11056 members.

11057 The **<netdb.h>** header shall define the following symbolic constants that evaluate to bitwise-
11058 distinct integer constants for use in the *ai_flags* field of the **addrinfo** structure:

11059 AI_PASSIVE	Socket address is intended for *bind*( ).
11060 AI_CANONNAME	Request for canonical name.
11061 AI_NUMERICHOST	Return numeric host address as name.
11062 AI_NUMERICSERV	Inhibit service name resolution.
11063 AI_V4MAPPED	If no IPv6 addresses are found, query for IPv4 addresses and return them 11064 to the caller as IPv4-mapped IPv6 addresses.
11065 AI_ALL	Query for both IPv4 and IPv6 addresses.
11066 AI_ADDRCONFIG	Query for IPv4 addresses only when an IPv4 address is configured; query 11067 for IPv6 addresses only when an IPv6 address is configured.

11068 The **<netdb.h>** header shall define the following symbolic constants that evaluate to bitwise-
11069 distinct integer constants for use in the *flags* argument to *getnameinfo*( ):

11070 NI_NOFQDN	Only the nodename portion of the FQDN is returned for local hosts.
11071 NI_NUMERICHOST	The numeric form of the node's address is returned instead of its name.
11072 NI_NAMEREQD	Return an error if the node's name cannot be located in the database.
11073 NI_NUMERICSERV	The numeric form of the service address is returned instead of its name.
11074 NI_NUMERICSCOPE	11075 For IPv6 addresses, the numeric form of the scope identifier is returned 11076 instead of its name.
11077 NI_DGRAM	Indicates that the service is a datagram service (SOCK_DGRAM).

Address Information Errors

The **<netdb.h>** header shall define the following symbolic constants for use as error values for *getaddrinfo*() and *getnameinfo*(). The values shall be suitable for use in **#if** preprocessing directives.

EAI_AGAIN	The name could not be resolved at this time. Future attempts may succeed.
EAI_BADFLAGS	The flags had an invalid value.
EAI_FAIL	A non-recoverable error occurred.
EAI_FAMILY	The address family was not recognized or the address length was invalid for the specified family.
EAI_MEMORY	There was a memory allocation failure.
EAI_NONAME	The name does not resolve for the supplied parameters.
	NI_NAMEREQD is set and the host's name cannot be located, or both *nodename* and *servname* were null.
EAI_SERVICE	The service passed was not recognized for the specified socket type.
EAI_SOCKTYPE	The intended socket type was not recognized.
EAI_SYSTEM	A system error occurred. The error code can be found in *errno*.
EAI_OVERFLOW	An argument buffer overflowed.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
void              endhostent(void);
void              endnetent(void);
void              endprotoent(void);
void              endservent(void);
void              freeaddrinfo(struct addrinfo *);
const char       *gai_strerror(int);
int               getaddrinfo(const char *restrict, const char *restrict,
                      const struct addrinfo *restrict,
                      struct addrinfo **restrict);
struct hostent   *gethostent(void);
int               getnameinfo(const struct sockaddr *restrict, socklen_t,
                      char *restrict, socklen_t, char *restrict,
                      socklen_t, int);
struct netent    *getnetbyaddr(uint32_t, int);
struct netent    *getnetbyname(const char *);
struct netent    *getnetent(void);
struct protoent  *getprotobyname(const char *);
struct protoent  *getprotobynumber(int);
struct protoent  *getprotoent(void);
struct servent   *getservbyname(const char *, const char *);
struct servent   *getservbyport(int, const char *);
struct servent   *getservent(void);
void              sethostent(int);
void              setnetent(int);
void              setprotoent(int);
```



```
void                setservent(int);
```

The **<netdb.h>** header shall define the **socklen_t** type through **typedef**, as described in **<sys/socket.h>**.

Inclusion of the **<netdb.h>** header may also make visible all symbols from **<netinet/in.h>**, **<sys/socket.h>**, and **<inttypes.h>**.

APPLICATION USAGE

The requirement that **addrinfo** does not include any additional members which have a floating-point type if an object of that type with an all-bits-zero representation does not have the value 0.0 is to allow initialization of an **addrinfo** *hints* structure (see XSH *freeaddrinfo*( )) using:

```
struct addrinfo hints;
memset(&hints, 0, sizeof hints);
```

as an alternative to the use of default initialization.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<inttypes.h>, **<netinet/in.h>**, **<sys/socket.h>**

XSH *bind*( ), *endhostent*( ), *endnetent*( ), *endprotoent*( ), *endservent*( ), *freeaddrinfo*( ), *gai_strerror*( ), *getnameinfo*( )

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The Open Group Base Resolution bwg2001-009 is applied, which changes the return type for *gai_strerror*( ) from **char** * to **const char** *. This is for coordination with the IPnG Working Group.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/11 is applied, adding a description of the NI_NUMERICSCOPE macro and correcting the *getnameinfo*( ) function prototype. These changes are for alignment with IPv6.

Issue 7

SD5-XBD-ERN-14 is applied, changing the description of the *s_port* member of the **servent** structure.

The obsolescent *h_errno* external integer, and the obsolescent *gethostbyaddr*( ) and *gethostbyname*( ) functions are removed, along with the HOST_NOT_FOUND, NO_DATA, NO_RECOVERY, and TRY_AGAIN macros.

This reference page is clarified with respect to macros and symbolic constants.

Issue 8

Austin Group Defect 940 is applied, requiring that the **addrinfo** structure does not include any additional members which have a floating-point type if an object of that type with an all-bits-zero representation does not have the value 0.0.

Austin Group Defect 1289 is applied, removing some redundant text from the DESCRIPTION.

Austin Group Defect 1327 is applied, changing ``flags'' to ``ai_flags''.

NAME

netinet/in.h — Internet address family

SYNOPSIS

```
#include <netinet/in.h>
```

DESCRIPTION

The **<netinet/in.h>** header shall define the following types:

in_port_t Equivalent to the type **uint16_t** as described in **<inttypes.h>**.

in_addr_t Equivalent to the type **uint32_t** as described in **<inttypes.h>**.

The **<netinet/in.h>** header shall define the **sa_family_t** type as described in **<sys/socket.h>**.

The **<netinet/in.h>** header shall define the **uint8_t** and **uint32_t** types as described in **<inttypes.h>**. Inclusion of the **<netinet/in.h>** header may also make visible all symbols from **<inttypes.h>** and **<sys/socket.h>**.

The **<netinet/in.h>** header shall define the **in_addr** structure, which shall include at least the following member:

```
in_addr_t  s_addr
```

The **<netinet/in.h>** header shall define the **sockaddr_in** structure, which shall include at least the following members:

`sa_family_t`	`sin_family`	AF_INET.
`in_port_t`	`sin_port`	Port number.
`struct in_addr`	`sin_addr`	IP address.

The *sin_port* and *sin_addr* members shall be in network byte order. If the *sin_port* value passed to *bind*() is zero, the port number bound to the socket shall be one chosen by the implementation from an implementation-defined port range to produce an unused local address.

The **sockaddr_in** structure is used to store addresses for the Internet address family. Pointers to this type shall be cast by applications to **struct sockaddr** * for use with socket functions.

IP6 The **<netinet/in.h>** header shall define the **in6_addr** structure, which shall include at least the following member:

```
uint8_t s6_addr[16]
```

This array is used to contain a 128-bit IPv6 address, stored in network byte order.

The **<netinet/in.h>** header shall define the **sockaddr_in6** structure, which shall include at least the following members:

`sa_family_t`	`sin6_family`	AF_INET6.
`in_port_t`	`sin6_port`	Port number.
`uint32_t`	`sin6_flowinfo`	IPv6 traffic class and flow information.
`struct in6_addr`	`sin6_addr`	IPv6 address.
`uint32_t`	`sin6_scope_id`	Set of interfaces for a scope.

The **sockaddr_in6** structure shall not include any additional members which have a floating-point type if an object of that type with an all-bits-zero representation does not have the value 0.0.

IP6 MX Implementations that define __STDC_IEC_559__ are required to treat the all-zero bit pattern for a floating point object as a representation of 0.0, and may therefore have floating-point type members.

IP6 The *sin6_port* and *sin6_addr* members shall be in network byte order. If the *sin6_port* value passed to *bind*( ) is zero, the port number bound to the socket shall be one chosen by the implementation from an implementation-defined port range to produce an unused local address.

Prior to calling a function in this standard which reads values from a **sockaddr_in6** structure (for example, *bind*( ) or *connect*( )), the application shall ensure that all members of the structure, including any additional non-standard members, if any, are initialized. If the **sockaddr_in6** structure has a non-standard member, and that member has a value other than the value that would result from default initialization, the behavior of any function in this standard that reads values from the **sockaddr_in6** structure is implementation-defined. All functions in this standard that return data in a **sockaddr_in6** structure (for example, *getaddrinfo*( ) or *accept*( )) shall initialize the structure in a way that meets the above requirements, and shall ensure that each non-standard member, if any, has a value that produces the same behavior as default initialization would in all functions in this standard which read values from a **sockaddr_in6** structure.

The *sin6_scope_id* field is a 32-bit integer that identifies a set of interfaces as appropriate for the scope of the address carried in the *sin6_addr* field. For a link scope *sin6_addr*, the application shall ensure that *sin6_scope_id* is a link index. For a site scope *sin6_addr*, the application shall ensure that *sin6_scope_id* is a site index. The mapping of *sin6_scope_id* to an interface or set of interfaces is implementation-defined.

The **<netinet/in.h>** header shall declare the following external variable:

```
const struct in6_addr in6addr_any
```

This variable is initialized by the system to contain the wildcard IPv6 address. The **<netinet/in.h>** header also defines the IN6ADDR_ANY_INIT macro. This macro shall be constant at compile time and can be used to initialize a variable of type **struct in6_addr** to the IPv6 wildcard address.

The **<netinet/in.h>** header shall declare the following external variable:

```
const struct in6_addr in6addr_loopback
```

This variable is initialized by the system to contain the loopback IPv6 address. The **<netinet/in.h>** header also defines the IN6ADDR_LOOPBACK_INIT macro. This macro shall be constant at compile time and can be used to initialize a variable of type **struct in6_addr** to the IPv6 loopback address.

The **<netinet/in.h>** header shall define the **ipv6_mreq** structure, which shall include at least the following members:

```
struct in6_addr  ipv6mr_multiaddr   IPv6 multicast address.
unsigned         ipv6mr_interface   Interface index.
```

The **<netinet/in.h>** header shall define the following symbolic constants for use as values of the *level* argument of *getsockopt*( ) and *setsockopt*( ):

IPPROTO_IP	Internet protocol.
IP6 IPPROTO_IPV6	Internet Protocol Version 6.
IPPROTO_ICMP	Control message protocol.

11246 RS IPPROTO_RAW Raw IP Packets Protocol.

11247 IPPROTO_TCP Transmission control protocol.

11248 IPPROTO_UDP User datagram protocol.

11249 The **<netinet/in.h>** header shall define the following symbolic constant for use as a local address
11250 in the structure passed to *bind*( ):

11251 INADDR_ANY IPv4 wildcard address.

11252 The **<netinet/in.h>** header shall define the following symbolic constant for use as a destination
11253 address in the structures passed to *connect*( ), *sendmsg*( ), and *sendto*( ):

11254 INADDR_BROADCAST IPv4 broadcast address.

11255 The **<netinet/in.h>** header shall define the following symbolic constant, with the value
11256 specified, to help applications declare buffers of the proper size to store IPv4 addresses in string
11257 form:

11258 INET_ADDRSTRLEN 16. Length of the string form for IP.

11259 The *htonl*( ), *htons*( ), *ntohl*( ), and *ntohs*( ) functions shall be available as described in
11260 **<arpa/inet.h>**. Inclusion of the **<netinet/in.h>** header may also make visible all symbols from
11261 **<arpa/inet.h>**.

11262 IP6 The **<netinet/in.h>** header shall define the following symbolic constant, with the value
11263 specified, to help applications declare buffers of the proper size to store IPv6 addresses in string
11264 form:

11265 INET6_ADDRSTRLEN 46. Length of the string form for IPv6.

11266 IP6 The **<netinet/in.h>** header shall define the following symbolic constants, with distinct integer
11267 values, for use in the *option_name* argument in the *getsockopt*( ) or *setsockopt*( ) functions at
11268 protocol level IPPROTO_IPV6:

11269 IPV6_JOIN_GROUP Join a multicast group.

11270 IPV6_LEAVE_GROUP Quit a multicast group.

11271 IPV6_MULTICAST_HOPS
11272 Multicast hop limit.

11273 IPV6_MULTICAST_IF Interface to use for outgoing multicast packets.

11274 IPV6_MULTICAST_LOOP
11275 Multicast packets are delivered back to the local application.

11276 IPV6_UNICAST_HOPS Unicast hop limit.

11277 IPV6_V6ONLY Restrict AF_INET6 socket to IPv6 communications only.

11278 The **<netinet/in.h>** header shall define the following macros that test for special IPv6 addresses.
11279 Each macro is of type **int** and takes a single argument of type **const struct in6_addr** *:

11280 IN6_IS_ADDR_UNSPECIFIED
11281 Unspecified address.

11282 IN6_IS_ADDR_LOOPBACK
11283 Loopback address.

IN6_IS_ADDR_MULTICAST
Multicast address.

IN6_IS_ADDR_LINKLOCAL
Unicast link-local address.

IN6_IS_ADDR_SITELOCAL
Unicast site-local address.

IN6_IS_ADDR_V4MAPPED
IPv4 mapped address.

IN6_IS_ADDR_V4COMPAT
IPv4-compatible address.

IN6_IS_ADDR_MC_NODELOCAL
Multicast node-local address.

IN6_IS_ADDR_MC_LINKLOCAL
Multicast link-local address.

IN6_IS_ADDR_MC_SITELOCAL
Multicast site-local address.

IN6_IS_ADDR_MC_ORGLOCAL
Multicast organization-local address.

IN6_IS_ADDR_MC_GLOBAL
Multicast global address.

APPLICATION USAGE

Although applications are required to initialize all members (including any non-standard ones) of a **sockaddr_in6** structure, the same is not required for the **sockaddr_in** structure, since historically many applications only initialized the standard members. Despite this, applications are encouraged to initialize **sockaddr_in** structures in a manner similar to the required initialization of **sockaddr_in6** structures.

The requirement that **sockaddr_in6** does not include any additional members which have a floating-point type if an object of that type with an all-bits-zero representation does not have the value 0.0 is to allow initialization of a **sockaddr_in6** structure using:

```
struct sockaddr_in6 sa;
memset(&sa, 0, sizeof sa);
```

as an alternative to the use of default initialization.

RATIONALE

The INADDR_ANY and INADDR_BROADCAST values are byte-order-neutral and thus their byte order is not specified. Many implementations have additional constants as extensions, such as INADDR_LOOPBACK, that are not byte-order-neutral. Traditionally, these constants are in host byte order, requiring the use of *htonl*( ) when using them in a **sockaddr_in** structure.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 4.13 (on page 99), **<arpa/inet.h>**, **<inttypes.h>**, **<sys/socket.h>**

XSH *bind*( ), *connect*( ), *getsockopt*( ), *htonl*( ), *sendmsg*( ), *sendto*( ), *setsockopt*( )

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The *sin_zero* member was removed from the **sockaddr_in** structure as per The Open Group Base Resolution bwg2001-004.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/12 is applied, adding **const** qualifiers to the *in6addr_any* and *in6addr_loopback* external variables.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/22 is applied, making it clear which structure members are in network byte order.

Issue 7

This reference page is clarified with respect to macros and symbolic constants.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0061 [355] is applied.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0065 [934], XBD/TC2-2008/0066 [952], XBD/TC2-2008/0067 [934], and XBD/TC2-2008/0068 [952] are applied.

Issue 8

Austin Group Defect 940 is applied, requiring that the **sockaddr_in6** structure does not include any additional members which have a floating-point type if an object of that type with an all-bits-zero representation does not have the value 0.0.

Austin Group Defect 1068 is applied, specifying how a *sin_port* or *sin6_port* value of zero is handled by *bind*( ).

Austin Group Defect 1299 is applied, changing **<netinet_in.h>** to **<netinet/in.h>**.

NAME

netinet/tcp.h — definitions for the Internet Transmission Control Protocol (TCP)

SYNOPSIS

```
#include <netinet/tcp.h>
```

DESCRIPTION

The **<netinet/tcp.h>** header shall define the following symbolic constant for use as a socket option at the IPPROTO_TCP level:

TCP_NODELAY Avoid coalescing of small segments.

The implementation need not allow the value of the option to be set via *setsockopt*( ) or retrieved via *getsockopt*( ).

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/socket.h>

XSH *getsockopt*( ), *setsockopt*( )

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7

This reference page is clarified with respect to macros and symbolic constants.

NAME

nl_types.h — data types

SYNOPSIS

```
#include <nl_types.h>
```

DESCRIPTION

The **<nl_types.h>** header shall define at least the following types:

nl_catd Used by the message catalog functions *catopen*( ), *catgets*( ), and *catclose*( ) to identify a catalog descriptor.

nl_item Used by *nl_langinfo*( ) to identify items of *langinfo* data. Values of objects of type **nl_item** are defined in **<langinfo.h>**.

The **<nl_types.h>** header shall define at least the following symbolic constants:

NL_SETD Used by *gencat* when no $*set* directive is specified in a message text source file. This constant can be passed as the value of *set_id* on subsequent calls to *catgets*( ) (that is, to retrieve messages from the default message set). The value of NL_SETD is implementation-defined.

NL_CAT_LOCALE Value that can be passed as the *oflag* argument to *catopen*( ) to request that message catalog selection depends on the *LC_MESSAGES* locale category, rather than directly on the *LANG* environment variable.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int       catclose(nl_catd);
char     *catgets(nl_catd, int, int, const char *);
nl_catd   catopen(const char *, int);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<langinfo.h>

XSH *catclose*( ), *catgets*( ), *catopen*( ), *nl_langinfo*( )

XCU *gencat*

CHANGE HISTORY

First released in Issue 2.

Issue 7

The **<nl_types.h>** header is moved from the XSI option to the Base.

This reference page is clarified with respect to macros and symbolic constants.

Issue 8

The description of NL_CAT_LOCALE is updated to eliminate the use of ``must''.

11409 **NAME**

11410 poll.h — definitions for the poll( ) function

11411 **SYNOPSIS**

11412 `#include <poll.h>`

11413 **DESCRIPTION**

11414 The **<poll.h>** header shall define the **pollfd** structure, which shall include at least the following
11415 members:

11416 `int fd` The following descriptor being polled.
11417 `short events` The input event flags (see below).
11418 `short revents` The output event flags (see below).

11419 The **<poll.h>** header shall define the following type through **typedef**:

11420 **nfds_t** An unsigned integer type used for the number of file descriptors.

11421 The implementation shall support one or more programming environments in which the width
11422 of **nfds_t** is no greater than the width of type **long**. The names of these programming
11423 environments can be obtained using the *confstr*( ) function or the *getconf* utility.

11424 The **<poll.h>** header shall define the **sigset_t** type as described in **<signal.h>**.

11425 The **<poll.h>** header shall define the **timespec** structure as described in **<time.h>**.

11426 The **<poll.h>** header shall define the following symbolic constants, zero or more of which may
11427 be OR'ed together to form the *events* or *revents* members in the **pollfd** structure:

11428 POLLIN Data other than high-priority data may be read without blocking.

11429 POLLRDNORM Normal data may be read without blocking.

11430 POLLRDBAND Priority data may be read without blocking.

11431 POLLPRI High priority data may be read without blocking.

11432 POLLOUT Normal data may be written without blocking.

11433 POLLWRNORM Equivalent to POLLOUT.

11434 POLLWRBAND Priority data may be written.

11435 POLLERR An error has occurred (*revents* only).

11436 POLLHUP Device has been disconnected (*revents* only).

11437 POLLNVAL Invalid *fd* member (*revents* only).

11438 The significance and semantics of normal, priority, and high-priority data are file and device-
11439 specific.

11440 The following shall be declared as functions and may also be defined as macros. Function
11441 prototypes shall be provided.

```
11442 int   poll(struct pollfd [], nfds_t, int);
11443 int   ppoll(struct pollfd [], nfds_t, const struct timespec *restrict,
11444           const sigset_t *restrict);
```

11445 Inclusion of the **<poll.h>** header may make visible all symbols from the headers **<signal.h>** and
11446 **<time.h>**.

11447 **APPLICATION USAGE**

11448 None.

11449 **RATIONALE**

11450 None.

11451 **FUTURE DIRECTIONS**

11452 None.

11453 **SEE ALSO**

11454 XSH *confstr*( ), *poll*( )

11455 XCU *getconf*

11456 **CHANGE HISTORY**

11457 First released in Issue 4, Version 2.

11458 **Issue 6**

11459 The description of the symbolic constants is updated to match the *poll*( ) function.

11460 Text related to STREAMS has been moved to the *poll*( ) reference page.

11461 A note is added to the DESCRIPTION regarding the significance and semantics of normal,
11462 priority, and high-priority data.

11463 **Issue 7**

11464 The **<poll.h>** header is moved from the XSI option to the Base.

11465 **Issue 8**

11466 Austin Group Defect 1263 is applied, adding *ppoll*( ), requiring **<poll.h>** to define **sigset_t** and
11467 **struct timespec**, and allowing **<poll.h>** to make visible all symbols from **<signal.h>** and
11468 **<time.h>**.

NAME

pthread.h — threads

SYNOPSIS

```
#include <pthread.h>
```

DESCRIPTION

The **<pthread.h>** header shall define the following symbolic constants:

PTHREAD_BARRIER_SERIAL_THREAD
PTHREAD_CANCEL_ASYNCHRONOUS
PTHREAD_CANCEL_ENABLE
PTHREAD_CANCEL_DEFERRED
PTHREAD_CANCEL_DISABLE
PTHREAD_CANCELED
PTHREAD_CREATE_DETACHED
PTHREAD_CREATE_JOINABLE
TPS PTHREAD_EXPLICIT_SCHED
PTHREAD_INHERIT_SCHED
PTHREAD_MUTEX_DEFAULT
PTHREAD_MUTEX_ERRORCHECK
PTHREAD_MUTEX_NORMAL
PTHREAD_MUTEX_RECURSIVE
PTHREAD_MUTEX_ROBUST
PTHREAD_MUTEX_STALLED
PTHREAD_ONCE_INIT
RPI | TPI PTHREAD_PRIO_INHERIT
MC1 PTHREAD_PRIO_NONE
RPP | TPP PTHREAD_PRIO_PROTECT
PTHREAD_PROCESS_SHARED
PTHREAD_PROCESS_PRIVATE
TPS PTHREAD_SCOPE_PROCESS
PTHREAD_SCOPE_SYSTEM

The **<pthread.h>** header shall define the following compile-time constant expressions valid as initializers for the following types:

Name	Initializer for Type
PTHREAD_COND_INITIALIZER	**pthread_cond_t**
PTHREAD_MUTEX_INITIALIZER	**pthread_mutex_t**
PTHREAD_RWLOCK_INITIALIZER	**pthread_rwlock_t**

The **<pthread.h>** header shall define the following compile-time constant expression, valid as an initializer for **pthread_t**, representing a value that shall not compare equal to the thread ID of any existing thread:

PTHREAD_NULL

The **<pthread.h>** header shall define the **pthread_attr_t**, **pthread_barrier_t**, **pthread_barrierattr_t**, **pthread_cond_t**, **pthread_condattr_t**, **pthread_key_t**, **pthread_mutex_t**, **pthread_mutexattr_t**, **pthread_once_t**, **pthread_rwlock_t**, **pthread_rwlockattr_t**, **pthread_spinlock_t**, and **pthread_t** types as described in **<sys/types.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.


```
11515 OB      int   pthread_atfork(void (*)(void), void (*)(void),
11516                   void(*)(void));
11517         int   pthread_attr_destroy(pthread_attr_t *);
11518         int   pthread_attr_getdetachstate(const pthread_attr_t *, int *);
11519         int   pthread_attr_getguardsize(const pthread_attr_t *restrict,
11520                   size_t *restrict);
11521 TPS     int   pthread_attr_getinheritsched(const pthread_attr_t *restrict,
11522                   int *restrict);
11523         int   pthread_attr_getschedparam(const pthread_attr_t *restrict,
11524                   struct sched_param *restrict);
11525 TPS     int   pthread_attr_getschedpolicy(const pthread_attr_t *restrict,
11526                   int *restrict);
11527         int   pthread_attr_getscope(const pthread_attr_t *restrict,
11528                   int *restrict);
11529 TSA TSS int   pthread_attr_getstack(const pthread_attr_t *restrict,
11530                   void **restrict, size_t *restrict);
11531 TSS     int   pthread_attr_getstacksize(const pthread_attr_t *restrict,
11532                   size_t *restrict);
11533         int   pthread_attr_init(pthread_attr_t *);
11534         int   pthread_attr_setdetachstate(pthread_attr_t *, int);
11535         int   pthread_attr_setguardsize(pthread_attr_t *, size_t);
11536 TPS     int   pthread_attr_setinheritsched(pthread_attr_t *, int);
11537         int   pthread_attr_setschedparam(pthread_attr_t *restrict,
11538                   const struct sched_param *restrict);
11539 TPS     int   pthread_attr_setschedpolicy(pthread_attr_t *, int);
11540         int   pthread_attr_setscope(pthread_attr_t *, int);
11541 TSA TSS int   pthread_attr_setstack(pthread_attr_t *, void *, size_t);
11542 TSS     int   pthread_attr_setstacksize(pthread_attr_t *, size_t);
11543         int   pthread_barrier_destroy(pthread_barrier_t *);
11544         int   pthread_barrier_init(pthread_barrier_t *restrict,
11545                   const pthread_barrierattr_t *restrict, unsigned);
11546         int   pthread_barrier_wait(pthread_barrier_t *);
11547         int   pthread_barrierattr_destroy(pthread_barrierattr_t *);
11548 TSH     int   pthread_barrierattr_getpshared(
11549                   const pthread_barrierattr_t *restrict, int *restrict);
11550         int   pthread_barrierattr_init(pthread_barrierattr_t *);
11551 TSH     int   pthread_barrierattr_setpshared(pthread_barrierattr_t *, int);
11552         int   pthread_cancel(pthread_t);
11553         int   pthread_cond_broadcast(pthread_cond_t *);
11554         int   pthread_cond_clockwait(pthread_cond_t *restrict,
11555                   pthread_mutex_t *restrict, clockid_t,
11556                   const struct timespec *restrict);
11557         int   pthread_cond_destroy(pthread_cond_t *);
11558         int   pthread_cond_init(pthread_cond_t *restrict,
11559                   const pthread_condattr_t *restrict);
11560         int   pthread_cond_signal(pthread_cond_t *);
11561         int   pthread_cond_timedwait(pthread_cond_t *restrict,
11562                   pthread_mutex_t *restrict, const struct timespec *restrict);
11563         int   pthread_cond_wait(pthread_cond_t *restrict,
11564                   pthread_mutex_t *restrict);
11565         int   pthread_condattr_destroy(pthread_condattr_t *);
11566         int   pthread_condattr_getclock(const pthread_condattr_t *restrict,
```



```
11567                   clockid_t *restrict);
11568 TSH       int   pthread_condattr_getpshared(const pthread_condattr_t *restrict,
11569                   int *restrict);
11570           int   pthread_condattr_init(pthread_condattr_t *);
11571           int   pthread_condattr_setclock(pthread_condattr_t *, clockid_t);
11572 TSH       int   pthread_condattr_setpshared(pthread_condattr_t *, int);
11573           int   pthread_create(pthread_t *restrict, const pthread_attr_t *restrict,
11574                   void *(*)(void*), void *restrict);
11575           int   pthread_detach(pthread_t);
11576           int   pthread_equal(pthread_t, pthread_t);
11577           _Noreturn void
11578                 pthread_exit(void *);
11579 TCT       int   pthread_getcpuclockid(pthread_t, clockid_t *);
11580 TPS       int   pthread_getschedparam(pthread_t, int *restrict,
11581                   struct sched_param *restrict);
11582           void *pthread_getspecific(pthread_key_t);
11583           int   pthread_join(pthread_t, void **);
11584           int   pthread_key_create(pthread_key_t *, void (*)(void*));
11585           int   pthread_key_delete(pthread_key_t);
11586           int   pthread_mutex_clocklock(pthread_mutex_t *restrict, clockid_t,
11587                   const struct timespec *restrict);
11588           int   pthread_mutex_consistent(pthread_mutex_t *);
11589           int   pthread_mutex_destroy(pthread_mutex_t *);
11590 RPP|TPP   int   pthread_mutex_getprioceiling(const pthread_mutex_t *restrict,
11591                   int *restrict);
11592           int   pthread_mutex_init(pthread_mutex_t *restrict,
11593                   const pthread_mutexattr_t *restrict);
11594           int   pthread_mutex_lock(pthread_mutex_t *);
11595 RPP|TPP   int   pthread_mutex_setprioceiling(pthread_mutex_t *restrict, int,
11596                   int *restrict);
11597           int   pthread_mutex_timedlock(pthread_mutex_t *restrict,
11598                   const struct timespec *restrict);
11599           int   pthread_mutex_trylock(pthread_mutex_t *);
11600           int   pthread_mutex_unlock(pthread_mutex_t *);
11601           int   pthread_mutexattr_destroy(pthread_mutexattr_t *);
11602 RPP|TPP   int   pthread_mutexattr_getprioceiling(
11603                   const pthread_mutexattr_t *restrict, int *restrict);
11604 MC1       int   pthread_mutexattr_getprotocol(const pthread_mutexattr_t *restrict,
11605                   int *restrict);
11606 TSH       int   pthread_mutexattr_getpshared(const pthread_mutexattr_t *restrict,
11607                   int *restrict);
11608           int   pthread_mutexattr_getrobust(const pthread_mutexattr_t *restrict,
11609                   int *restrict);
11610           int   pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict,
11611                   int *restrict);
11612           int   pthread_mutexattr_init(pthread_mutexattr_t *);
11613 RPP|TPP   int   pthread_mutexattr_setprioceiling(pthread_mutexattr_t *, int);
11614 MC1       int   pthread_mutexattr_setprotocol(pthread_mutexattr_t *, int);
11615 TSH       int   pthread_mutexattr_setpshared(pthread_mutexattr_t *, int);
11616           int   pthread_mutexattr_setrobust(pthread_mutexattr_t *, int);
11617           int   pthread_mutexattr_settype(pthread_mutexattr_t *, int);
11618           int   pthread_once(pthread_once_t *, void (*)(void));
```



```
11619          int   pthread_rwlock_destroy(pthread_rwlock_t *);
11620          int   pthread_rwlock_init(pthread_rwlock_t *restrict,
11621                    const pthread_rwlockattr_t *restrict);
11622          int   pthread_rwlock_clockrdlock(pthread_rwlock_t *restrict,
11623                    clockid_t, const struct timespec *restrict);
11624          int   pthread_rwlock_clockwrlock(pthread_rwlock_t *restrict,
11625                    clockid_t, const struct timespec *restrict);
11626          int   pthread_rwlock_rdlock(pthread_rwlock_t *);
11627          int   pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict,
11628                    const struct timespec *restrict);
11629          int   pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict,
11630                    const struct timespec *restrict);
11631          int   pthread_rwlock_tryrdlock(pthread_rwlock_t *);
11632          int   pthread_rwlock_trywrlock(pthread_rwlock_t *);
11633          int   pthread_rwlock_unlock(pthread_rwlock_t *);
11634          int   pthread_rwlock_wrlock(pthread_rwlock_t *);
11635          int   pthread_rwlockattr_destroy(pthread_rwlockattr_t *);
11636   TSH    int   pthread_rwlockattr_getpshared(
11637                    const pthread_rwlockattr_t *restrict, int *restrict);
11638          int   pthread_rwlockattr_init(pthread_rwlockattr_t *);
11639   TSH    int   pthread_rwlockattr_setpshared(pthread_rwlockattr_t *, int);
11640          pthread_t
11641                pthread_self(void);
11642          int   pthread_setcancelstate(int, int *);
11643          int   pthread_setcanceltype(int, int *);
11644   TPS    int   pthread_setschedparam(pthread_t, int,
11645                    const struct sched_param *);
11646          int   pthread_setschedprio(pthread_t, int);
11647          int   pthread_setspecific(pthread_key_t, const void *);
11648          int   pthread_spin_destroy(pthread_spinlock_t *);
11649          int   pthread_spin_init(pthread_spinlock_t *, int);
11650          int   pthread_spin_lock(pthread_spinlock_t *);
11651          int   pthread_spin_trylock(pthread_spinlock_t *);
11652          int   pthread_spin_unlock(pthread_spinlock_t *);
11653          void  pthread_testcancel(void);
```

11654 The following may be declared as functions, or defined as macros, or both. If functions are
11655 declared, function prototypes shall be provided.

11656 *pthread_cleanup_pop*( )
11657 *pthread_cleanup_push*( )

11658 Inclusion of the **<pthread.h>** header shall make symbols defined in the headers **<sched.h>** and
11659 **<time.h>** visible.

APPLICATION USAGE

None.

RATIONALE

Since **pthread_t** is an opaque type, a definition of PTHREAD_NULL was added to allow for a null value of that type. Some conforming definitions of PTHREAD_NULL could be:

For a pointer type:

```
#define PTHREAD_NULL ((pthread_t)NULL)
```

For an integer type:

```
#define PTHREAD_NULL ((pthread_t)-42)
```

For a struct type:

```
#define PTHREAD_NULL ((const pthread_t){ .__foo = -1 })
```

FUTURE DIRECTIONS

None.

SEE ALSO

<sched.h>, **<sys/types.h>**, **<time.h>**

XSH *pthread_atfork*( ), *pthread_attr_destroy*( ), *pthread_attr_getdetachstate*( ), *pthread_attr_getguardsize*( ), *pthread_attr_getinheritsched*( ), *pthread_attr_getschedparam*( ), *pthread_attr_getschedpolicy*( ), *pthread_attr_getscope*( ), *pthread_attr_getstack*( ), *pthread_attr_getstacksize*( ), *pthread_barrier_destroy*( ), *pthread_barrier_wait*( ), *pthread_barrierattr_destroy*( ), *pthread_barrierattr_getpshared*( ), *pthread_cancel*( ), *pthread_cleanup_pop*( ), *pthread_cond_broadcast*( ), *pthread_cond_clockwait*( ), *pthread_cond_destroy*( ), *pthread_condattr_destroy*( ), *pthread_condattr_getclock*( ), *pthread_condattr_getpshared*( ), *pthread_create*( ), *pthread_detach*( ), *pthread_equal*( ), *pthread_exit*( ), *pthread_getcpuclockid*( ), *pthread_getschedparam*( ), *pthread_getspecific*( ), *pthread_join*( ), *pthread_key_create*( ), *pthread_key_delete*( ), *pthread_mutex_clocklock*( ), *pthread_mutex_consistent*( ), *pthread_mutex_destroy*( ), *pthread_mutex_getprioceiling*( ), *pthread_mutex_lock*( ), *pthread_mutexattr_destroy*( ), *pthread_mutexattr_getprioceiling*( ), *pthread_mutexattr_getprotocol*( ), *pthread_mutexattr_getpshared*( ), *pthread_mutexattr_getrobust*( ), *pthread_mutexattr_gettype*( ), *pthread_once*( ), *pthread_rwlock_clockrdlock*( ), *pthread_rwlock_clockwrlock*( ), *pthread_rwlock_destroy*( ), *pthread_rwlock_rdlock*( ), *pthread_rwlock_trywrlock*( ), *pthread_rwlock_unlock*( ), *pthread_rwlockattr_destroy*( ), *pthread_rwlockattr_getpshared*( ), *pthread_self*( ), *pthread_setcancelstate*( ), *pthread_setschedprio*( ), *pthread_spin_destroy*( ), *pthread_spin_lock*( ), *pthread_spin_unlock*( )

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The RTT margin markers are broken out into their POSIX options.

The Open Group Corrigendum U021/9 is applied, correcting the prototype for the *pthread_cond_wait*( ) function.

The Open Group Corrigendum U026/2 is applied, correcting the prototype for the *pthread_setschedparam*( ) function so that its second argument is of type **int**.

The *pthread_getcpuclockid*( ) and *pthread_mutex_timedlock*( ) functions are added for alignment with IEEE Std 1003.1d-1999.

The following functions are added for alignment with IEEE Std 1003.1j-2000:

pthread_barrier_destroy( ), *pthread_barrier_init*( ), *pthread_barrier_wait*( ), *pthread_barrierattr_destroy*( ), *pthread_barrierattr_getpshared*( ), *pthread_barrierattr_init*( ), *pthread_barrierattr_setpshared*( ), *pthread_condattr_getclock*( ), *pthread_condattr_setclock*( ), *pthread_rwlock_timedrdlock*( ), *pthread_rwlock_timedwrlock*( ), *pthread_spin_destroy*( ), *pthread_spin_init*( ), *pthread_spin_lock*( ), *pthread_spin_trylock*( ), and *pthread_spin_unlock*( ).

PTHREAD_RWLOCK_INITIALIZER is removed for alignment with IEEE Std 1003.1j-2000.

Functions previously marked as part of the Read-Write Locks option are now moved to the Threads option.

The **restrict** keyword is added to the prototypes for *pthread_attr_getguardsize*( ), *pthread_attr_getinheritsched*( ), *pthread_attr_getschedparam*( ), *pthread_attr_getschedpolicy*( ), *pthread_attr_getscope*( ), *pthread_attr_getstackaddr*,( ) *pthread_attr_getstacksize*( ), *pthread_attr_setschedparam*( ), *pthread_barrier_init*( ), *pthread_barrierattr_getpshared*( ), *pthread_cond_init*( ), *pthread_cond_signal*( ), *pthread_cond_timedwait*( ), *pthread_cond_wait*( ), *pthread_condattr_getclock*( ), *pthread_condattr_getpshared*( ), *pthread_create*( ), *pthread_getschedparam*( ), *pthread_mutex_getprioceiling*( ), *pthread_mutex_init*( ), *pthread_mutex_setprioceiling*( ), *pthread_mutexattr_getprioceiling*( ), *pthread_mutexattr_getprotocol*( ), *pthread_mutexattr_getpshared*( ), *pthread_mutexattr_gettype*( ), *pthread_rwlock_init*( ), *pthread_rwlock_timedrdlock*( ), *pthread_rwlock_timedwrlock*( ), *pthread_rwlockattr_getpshared*( ), and *pthread_sigmask*( ).

IEEE PASC Interpretation 1003.1 #86 is applied, allowing the symbols from **<sched.h>** and **<time.h>** to be made visible when **<pthread.h>** is included. Previously this was an XSI option.

IEEE PASC Interpretation 1003.1c #42 is applied, removing the requirement for prototypes for the *pthread_kill*( ) and *pthread_sigmask*( ) functions. These are required to be in the **<signal.h>** header. They are allowed here through the name space rules.

IEEE PASC Interpretation 1003.1 #96 is applied, adding the *pthread_setschedprio*( ) function.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/13 is applied, correcting shading errors that were in contradiction with the System Interfaces volume of POSIX.1-2024.

Issue 7

SD5-XBD-ERN-55 is applied, adding the **restrict** keyword to the *pthread_mutex_timedlock*( ) function prototype.

SD5-XBD-ERN-62 is applied.

Austin Group Interpretation 1003.1-2001 #048 is applied, reinstating the PTHREAD_RWLOCK_INITIALIZER symbol.

The **<pthread.h>** header is moved from the Threads option to the Base.

The following extended mutex types are moved from the XSI option to the Base:

PTHREAD_MUTEX_NORMAL
PTHREAD_MUTEX_ERRORCHECK
PTHREAD_MUTEX_RECURSIVE
PTHREAD_MUTEX_DEFAULT

The PTHREAD_MUTEX_ROBUST and PTHREAD_MUTEX_STALLED symbols and the *pthread_mutex_consistent*( ), *pthread_mutexattr_getrobust*( ), and *pthread_mutexattr_setrobust*( ) functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Functionality relating to the Thread Priority Protection and Thread Priority Inheritance options is changed to be Non-Robust Mutex or Robust Mutex Priority Protection and Non-Robust Mutex

or Robust Mutex Priority Inheritance, respectively.

This reference page is clarified with respect to macros and symbolic constants.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0069 [624] is applied.

Issue 8

Austin Group Defect 599 is applied, adding the PTHREAD_NULL constant.

Austin Group Defect 851 is applied, marking *pthread_atfork*( ) as obsolescent.

Austin Group Defect 1216 is applied, adding *pthread_cond_clockwait*( ), *pthread_mutex_clocklock*( ), *pthread_rwlock_clockrdlock*( ), and *pthread_rwlock_clockwrlock*( ).

Austin Group Defect 1302 is applied, adding `_Noreturn` to *pthread_exit*( ).

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

NAME

pwd.h — password structure

SYNOPSIS

```
#include <pwd.h>
```

DESCRIPTION

The **<pwd.h>** header shall define the **struct passwd**, structure, which shall include at least the following members:

```
char     *pw_name    User's login name.
uid_t     pw_uid     Numerical user ID.
gid_t     pw_gid     Numerical group ID.
char     *pw_dir     Initial working directory.
char     *pw_shell   Program to use as shell.
```

The **<pwd.h>** header shall define the **gid_t**, **uid_t**, and **size_t** types as described in **<sys/types.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

XSI
```
void           endpwent(void);
struct passwd *getpwent(void);
struct passwd *getpwnam(const char *);
int            getpwnam_r(const char *, struct passwd *, char *,
                   size_t, struct passwd **);
struct passwd *getpwuid(uid_t);
int            getpwuid_r(uid_t, struct passwd *, char *,
                   size_t, struct passwd **);
void           setpwent(void);
```
XSI

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>

XSH *endpwent*( ), *getpwnam*( ), *getpwuid*( )

CHANGE HISTORY

First released in Issue 1.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **gid_t** and **uid_t** types are mandated.

- The *getpwnam_r*() and *getpwuid_r*() functions are marked as part of the Thread-Safe Functions option.

Issue 7

SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

11804 **NAME**

11805 regex.h — regular expression matching types

11806 **SYNOPSIS**

11807 `#include <regex.h>`

11808 **DESCRIPTION**

11809 The **<regex.h>** header shall define the structures and symbolic constants used by the *regcomp*( ),
11810 *regexec*( ), *regerror*( ), and *regfree*( ) functions.

11811 The **<regex.h>** header shall define the **regex_t** structure type, which shall include at least the
11812 following member:

11813 `size_t re_nsub` Number of parenthesized subexpressions.

11814 The **<regex.h>** header shall define the **size_t** type as described in **<sys/types.h>**.

11815 The **<regex.h>** header shall define the **regoff_t** type as a signed integer type that can hold the
11816 largest value that can be stored in either a **ptrdiff_t** type or a **ssize_t** type.

11817 The **<regex.h>** header shall define the **regmatch_t** structure type, which shall include at least the
11818 following members:

11819 `regoff_t rm_so` Byte offset from start of string
11820 to start of substring.
11821 `regoff_t rm_eo` Byte offset from start of string of the
11822 first character after the end of substring.

11823 The **<regex.h>** header shall define the following symbolic constants for the *cflags* parameter to
11824 the *regcomp*( ) function:

11825 REG_EXTENDED Use Extended Regular Expressions.

11826 REG_ICASE Perform pattern matching in a case-insensitive manner.

11827 REG_MINIMAL Change default matching behavior to leftmost shortest possible match.
11828 Only applicable to REG_EXTENDED regular expressions.

11829 REG_NOSUB Report only success or fail in *regexec*( ).

11830 REG_NEWLINE Change the handling of <newline>.

11831 The **<regex.h>** header shall define the following symbolic constants for the *eflags* parameter to
11832 the *regexec*( ) function:

11833 REG_NOTBOL The <circumflex> character (`'^'`), when taken as a special character, does
11834 not match the beginning of *string*.

11835 REG_NOTEOL The <dollar-sign> (`'$'`), when taken as a special character, does not
11836 match the end of *string*.

11837 The **<regex.h>** header shall define the following symbolic constants as error return values:

11838 REG_NOMATCH *regexec*( ) failed to match.

11839 REG_BADPAT Invalid regular expression.

11840 REG_ECOLLATE Invalid collating element referenced.

11841 REG_ECTYPE Invalid character class type referenced.

11842 REG_EESCAPE Trailing <backslash> character in pattern.

REG_ESUBREG	Number in *digit* invalid or in error.
REG_EBRACK	`"[]"` imbalance.
REG_EPAREN	`"\(\)"` or `"()"` imbalance.
REG_EBRACE	`"\{\}"` imbalance.
REG_BADBR	Content of `"\{\}"` invalid: not a number, number too large, more than two numbers, first larger than second.
REG_ERANGE	Invalid endpoint in range expression.
REG_ESPACE	Out of memory.
REG_BADRPT	`'?'`, `'*'`, or `'+'` not preceded by valid regular expression.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int    regcomp(regex_t *restrict, const char *restrict, int);
size_t regerror(int, const regex_t *restrict, char *restrict, size_t);
int    regexec(const regex_t *restrict, const char *restrict, size_t,
           regmatch_t [restrict], int);
void   regfree(regex_t *);
```

The implementation may define additional macros or constants using names beginning with REG_.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>

XSH *regcomp*( )

CHANGE HISTORY

First released in Issue 4.

Originally derived from the ISO POSIX-2 standard.

Issue 6

The REG_ENOSYS constant is marked obsolescent.

The **restrict** keyword is added to the prototypes for *regcomp*( ), *regerror*( ), and *regexec*( ).

A statement is added that the **size_t** type is defined as described in **<sys/types.h>**.

Issue 7

SD5-XBD-ERN-60 is applied.

The obsolescent REG_ENOSYS constant is removed.

This reference page is clarified with respect to macros and symbolic constants.

Issue 8

Austin Group Defect 793 is applied, adding REG_MINIMAL.

Austin Group Defect 1031 is applied, changing the description of REG_ICASE.

NAME

sched.h — execution scheduling

SYNOPSIS

```
#include <sched.h>
```

DESCRIPTION

PS The **<sched.h>** header shall define the **pid_t** type as described in **<sys/types.h>**.

SS|TSP The **<sched.h>** header shall define the **time_t** type as described in **<sys/types.h>**.

The **<sched.h>** header shall define the **timespec** structure as described in **<time.h>**.

The **<sched.h>** header shall define the **sched_param** structure, which shall include the scheduling parameters required for implementation of each supported scheduling policy. This structure shall include at least the following member:

`int`	`sched_priority`	Process or thread execution scheduling priority.

SS|TSP The **sched_param** structure defined in **<sched.h>** shall include the following members in addition to those specified above:

`int`	`sched_ss_low_priority`	Low scheduling priority for sporadic server.
`struct timespec`	`sched_ss_repl_period`	Replenishment period for sporadic server.
`struct timespec`	`sched_ss_init_budget`	Initial budget for sporadic server.
`int`	`sched_ss_max_repl`	Maximum pending replenishments for sporadic server.

Each process or thread is controlled by an associated scheduling policy and priority. Associated with each policy is a priority range. Each policy definition specifies the minimum priority range for that policy. The priority ranges for each policy may overlap the priority ranges of other policies.

Four scheduling policies are defined; others may be defined by the implementation. The four standard policies are indicated by the values of the following symbolic constants:

PS|TPS SCHED_FIFO First in-first out (FIFO) scheduling policy.

PS|TPS SCHED_RR Round robin scheduling policy.

SS|TSP SCHED_SPORADIC Sporadic server scheduling policy.

PS|TPS SCHED_OTHER Another scheduling policy.

The values of these constants are distinct.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

PS|TPS
```
int    sched_get_priority_max(int);
int    sched_get_priority_min(int);
PS
int    sched_getparam(pid_t, struct sched_param *);
int    sched_getscheduler(pid_t);
PS|TPS
int    sched_rr_get_interval(pid_t, struct timespec *);
PS
int    sched_setparam(pid_t, const struct sched_param *);
int    sched_setscheduler(pid_t, int, const struct sched_param *);
int    sched_yield(void);
```


Inclusion of the **<sched.h>** header may make visible all symbols from the **<time.h>** header.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, **<time.h>**

XSH *sched_get_priority_max*( ), *sched_getparam*( ), *sched_getscheduler*( ), *sched_rr_get_interval*( ), *sched_setparam*( ), *sched_setscheduler*( ), *sched_yield*( )

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The **<sched.h>** header is marked as part of the Process Scheduling option.

Sporadic server members are added to the **sched_param** structure, and the SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #108 is applied, correcting the **sched_param** structure whose members *sched_ss_repl_period* and *sched_ss_init_budget* should be type **struct timespec** and not **timespec**.

Symbols from **<time.h>** may be made visible when **<sched.h>** is included.

IEEE Std 1003.1-2001/Cor 1-2002, items XSH/TC1/D6/52 and XSH/TC1/D6/53 are applied, aligning the function prototype shading and margin codes with the System Interfaces volume of IEEE Std 1003.1-2001.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/23 is applied, updating the DESCRIPTION to differentiate between thread and process execution.

Issue 7

SD5-XBD-ERN-13 is applied.

Austin Group Interpretation 1003.1-2001 #064 is applied, correcting the options markings.

The **<sched.h>** headers is moved from the Threads option to the Base.

Declarations for the **pid_t** and **time_t** types and the **timespec** structure are added.

NAME

search.h — search tables

SYNOPSIS

XSI
```
#include <search.h>
```

DESCRIPTION

The **<search.h>** header shall define the **ENTRY** type for structure **entry** which shall include the following members:

```
char    *key
void    *data
```

and shall define **ACTION** and **VISIT** as enumeration data types through type definitions as follows:

```
enum { FIND, ENTER } ACTION;
enum { preorder, postorder, endorder, leaf } VISIT;
```

The **<search.h>** header shall define the **size_t** type as described in **<sys/types.h>**.

The **<search.h>** header shall define via **typedef** the **posix_tnode** type as an alias for **void**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int           hcreate(size_t);
void          hdestroy(void);
ENTRY        *hsearch(ENTRY, ACTION);
void          insque(void *, void *);
void         *lfind(const void *, const void *, size_t *,
                  size_t, int (*)(const void *, const void *));
void         *lsearch(const void *, void *, size_t *,
                  size_t, int (*)(const void *, const void *));
void          remque(void *);
void         *tdelete(const void *restrict, posix_tnode **restrict,
                  int(*)(const void *, const void *));
posix_tnode *tfind(const void *, posix_tnode *const *,
                  int(*)(const void *, const void *));
posix_tnode *tsearch(const void *, posix_tnode **,
                  int(*)(const void *, const void *));
void          twalk(const posix_tnode *,
                  void (*)(const posix_tnode *, VISIT, int));
```

APPLICATION USAGE

None.

RATIONALE

Earlier versions of this standard explicitly used **void** for both node and key references where this version now uses **posix_tnode** for nodes and keeps **void** in the text referring only to keys. In order to preserve backwards compatibility, this version defines **posix_tnode** as an alias for **void**. The change was made to make the function prototypes more easily understandable.

FUTURE DIRECTIONS

None.

12000 **SEE ALSO**

12001 **<sys/types.h>**

12002 XSH *hcreate*( ), *insque*( ), *lsearch*( ), *tdelete*( )

12003 **CHANGE HISTORY**

12004 First released in Issue 1. Derived from Issue 1 of the SVID.

12005 **Issue 6**

12006 The Open Group Corrigendum U021/6 is applied, updating the prototypes for *tdelete*( ) and
12007 *tsearch*( ).

12008 The **restrict** keyword is added to the prototype for *tdelete*( ).

12009 **Issue 8**

12010 Austin Group Defect 1011 is applied, adding the **posix_tnode** type and changing some
12011 prototypes to use it instead of **void**.

12012 **NAME**

12013 semaphore.h — semaphores

12014 **SYNOPSIS**

12015 `#include <semaphore.h>`

12016 **DESCRIPTION**

12017 The **<semaphore.h>** header shall define the **sem_t** type, used in performing semaphore
12018 operations. The semaphore may be implemented using a file descriptor, in which case
12019 applications are able to open up at least a total of {OPEN_MAX} files and semaphores.

12020 The **<semaphore.h>** header shall define the **timespec** structure as described in **<time.h>**.

12021 The **<semaphore.h>** header shall define the symbolic constant SEM_FAILED which shall have
12022 type **sem_t** *.

12023 The **<semaphore.h>** header shall define O_CREAT and O_EXCL as described in **<fcntl.h>**.

12024 The following shall be declared as functions and may also be defined as macros. Function
12025 prototypes shall be provided.

```
12026 int    sem_clockwait(sem_t *restrict, clockid_t,
12027            const struct timespec *restrict);
12028 int    sem_close(sem_t *);
12029 int    sem_destroy(sem_t *);
12030 int    sem_getvalue(sem_t *restrict, int *restrict);
12031 int    sem_init(sem_t *, int, unsigned);
12032 sem_t *sem_open(const char *, int, ...);
12033 int    sem_post(sem_t *);
12034 int    sem_timedwait(sem_t *restrict, const struct timespec *restrict);
12035 int    sem_trywait(sem_t *);
12036 int    sem_unlink(const char *);
12037 int    sem_wait(sem_t *);
```

12038 Inclusion of the **<semaphore.h>** header may make visible symbols defined in the **<fcntl.h>** and
12039 **<time.h>** headers.

12040 **APPLICATION USAGE**

12041 None.

12042 **RATIONALE**

12043 None.

12044 **FUTURE DIRECTIONS**

12045 None.

12046 **SEE ALSO**

12047 **<fcntl.h>**, **<sys/types.h>**, **<time.h>**

12048 XSH *sem_clockwait*( ), *sem_close*( ), *sem_destroy*( ), *sem_getvalue*( ), *sem_init*( ), *sem_open*( ),
12049 *sem_post*( ), *sem_trywait*( ), *sem_unlink*( )

12050 **CHANGE HISTORY**

12051 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

12052 **Issue 6**

12053 The **<semaphore.h>** header is marked as part of the Semaphores option.

12054 The Open Group Corrigendum U021/3 is applied, adding a description of SEM_FAILED.

12055 The *sem_timedwait*( ) function is added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the prototypes for *sem_getvalue*( ) and *sem_timedwait*( ).

Issue 7

SD5-XBD-ERN-57 is applied, allowing the header to make visible symbols from the **<time.h>** header.

The **<semaphore.h>** header is moved from the Semaphores option to the Base.

This reference page is clarified with respect to macros and symbolic constants.

Issue 8

Austin Group Defect 592 is applied, requiring **<semaphore.h>** to define the **timespec** structure.

Austin Group Defect 593 is applied, adding O_CREAT and O_EXCL.

Austin Group Defect 1216 is applied, adding *sem_clockwait*( ).

NAME

setjmp.h — stack environment declarations

SYNOPSIS

```
#include <setjmp.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to enable the visibility of these symbols in this header.

CX The **<setjmp.h>** header shall define the array types **jmp_buf** and **sigjmp_buf**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
_Noreturn void   longjmp(jmp_buf, int);
CX _Noreturn void   siglongjmp(sigjmp_buf, int);
```

The following may be declared as functions, or defined as macros, or both. If functions are declared, function prototypes shall be provided.

```
int    setjmp(jmp_buf);
CX int    sigsetjmp(sigjmp_buf, int);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH Section 2.2 (on page 496), *longjmp*( ), *setjmp*( ), *siglongjmp*( ), *sigsetjmp*( )

CHANGE HISTORY

First released in Issue 1.

Issue 6

Extensions beyond the ISO C standard are marked.

Issue 7

SD5-XBD-ERN-6 is applied.

Issue 8

Austin Group Defect 1302 is applied, adding `_Noreturn` to *longjmp*( ) and *siglongjmp*( ).

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

NAME

signal.h — signals

SYNOPSIS

```
#include <signal.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to enable the visibility of these symbols in this header.

The **<signal.h>** header shall define the following macros, which shall expand to constant expressions with distinct values that have a type compatible with the second argument to, and the return value of, the *signal*() function, and whose values shall compare unequal to the address of any declarable function.

SIG_DFL	Request for default signal handling.
SIG_ERR	Return value from *signal*() in case of error.
SIG_IGN	Request that signal be ignored.

CX The **<signal.h>** header shall define the **pthread_t**, **size_t**, and **uid_t** types as described in **<sys/types.h>**.

The **<signal.h>** header shall define the **timespec** structure as described in **<time.h>**.

The **<signal.h>** header shall define the following data types:

sig_atomic_t	Possibly volatile-qualified integer type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts.
CX **sigset_t**	Integer or structure type of an object used to represent sets of signals.
CX **pid_t**	As described in **<sys/types.h>**.

CX The **<signal.h>** header shall define the **pthread_attr_t** type as described in **<sys/types.h>**.

The **<signal.h>** header shall define the **sigevent** structure, which shall include at least the following members:

`int`	`sigev_notify`	Notification type.
`int`	`sigev_signo`	Signal number.
`union sigval`	`sigev_value`	Signal value.
`void`	`(*sigev_notify_function)(union sigval)`	Notification function.
`pthread_attr_t *`	`sigev_notify_attributes`	Notification attributes.

The **<signal.h>** header shall define the following symbolic constants for the values of *sigev_notify*:

SIGEV_NONE	No asynchronous notification is delivered when the event of interest occurs.
SIGEV_SIGNAL	A queued signal, with an application-defined value, is generated when the event of interest occurs.
SIGEV_THREAD	A notification function is called to perform notification.

The **sigval** union shall be defined as:

```
int     sival_int     Integer signal value.
void   *sival_ptr     Pointer signal value.
```

The **<signal.h>** header shall declare the SIGRTMIN and SIGRTMAX macros, which shall expand to positive integer expressions with type **int**, but which need not be constant expressions. These macros specify a range of signal numbers that are reserved for application use and for which the realtime signal behavior specified in this volume of POSIX.1-2024 is supported. The signal numbers in this range do not overlap any of the signals specified in the following table.

The range SIGRTMIN through SIGRTMAX inclusive shall include at least {RTSIG_MAX} signal numbers. The value of SIGRTMAX shall be less than the value returned by *sysconf* (_SC_NSIG).

It is implementation-defined whether realtime signal behavior is supported for other signals.

The **<signal.h>** header shall define the following symbolic constant. The value shall be suitable for use in **#if** preprocessing directives:

SIG2STR_MAX — Maximum size of a signal name returned by *sig2str*( ), including the terminating null byte.

The **<signal.h>** header shall define the following macros that are used to refer to the signals that occur in the system. Signals defined here begin with the letters SIG followed by an uppercase letter. The macros shall expand to positive integer constant expressions with type **int** and
CX distinct values less than the value of {NSIG_MAX} defined in **<limits.h>**. The value 0 is reserved for use as the null signal (see *kill*( )). Additional implementation-defined signals may occur in the system.

The ISO C standard only requires the signal names SIGABRT, SIGFPE, SIGILL, SIGINT, SIGSEGV, and SIGTERM to be defined. An implementation need not generate any of these six
CX signals, except as a result of explicit use of interfaces that generate signals, such as *raise*( ), *kill*( ), the General Terminal Interface (see Section 11.1.9, on page 203), and the *kill* utility, unless otherwise stated (see, for example, XSH Section 2.8.3.3, on page 530).

The following signals shall be supported on all implementations (default actions are explained below the table):

Signal	Default Action	Description
SIGABRT	A	Process abort signal.
SIGALRM	T	Alarm clock.
SIGBUS	A	Access to an undefined portion of a memory object.
SIGCHLD	I	Child process terminated, stopped,
XSI		or continued.
SIGCONT	C	Continue executing, if stopped.
SIGFPE	A	Erroneous arithmetic operation.
SIGHUP	T	Hangup.
SIGILL	A	Illegal instruction.
SIGINT	T	Terminal interrupt signal.
SIGKILL	T	Kill (cannot be caught or ignored).
SIGPIPE	T	Write on a pipe with no one to read it.
SIGQUIT	A	Terminal quit signal.
SIGSEGV	A	Invalid memory reference.
SIGSTOP	S	Stop executing (cannot be caught or ignored).
SIGTERM	T	Termination signal.
SIGTSTP	S	Terminal stop signal.
SIGTTIN	S	Background process attempting read.
SIGTTOU	S	Background process attempting write.
SIGUSR1	T	User-defined signal 1.
SIGUSR2	T	User-defined signal 2.
SIGWINCH	I	Terminal window size changed.
XSI SIGSYS	A	Bad system call.
SIGTRAP	A	Trace/breakpoint trap.
SIGURG	I	High bandwidth data is available at a socket.
XSI SIGVTALRM	T	Virtual timer expired.
SIGXCPU	A	CPU time limit exceeded.
SIGXFSZ	A	File size limit exceeded.

The default actions are as follows:

T Abnormal termination of the process.
A Abnormal termination of the process with additional actions.
I Ignore the signal.
S Stop the process.
C Continue the process, if it is stopped; otherwise, ignore the signal.

The effects on the process in each case are described in XSH Section 2.4.3 (on page 516).

CX The **<signal.h>** header shall declare the **sigaction** structure, which shall include at least the following members:

```
void    (*sa_handler)(int)   Pointer to a signal-catching function
                             or one of the SIG_IGN or SIG_DFL.
sigset_t sa_mask             Set of signals to be blocked during execution
                             of the signal handling function.
int      sa_flags            Special flags.
void    (*sa_sigaction)(int, siginfo_t *, void *)
                             Pointer to a signal-catching function.
```


12212 CX The storage occupied by *sa_handler* and *sa_sigaction* may overlap, and a conforming application
12213 shall not use both simultaneously.

12214 The **<signal.h>** header shall define the following macros which shall expand to integer constant
12215 expressions that need not be usable in **#if** preprocessing directives:

12216 CX SIG_BLOCK The resulting set is the union of the current set and the signal set pointed
12217 to by the argument *set*.

12218 CX SIG_UNBLOCK The resulting set is the intersection of the current set and the complement
12219 of the signal set pointed to by the argument *set*.

12220 CX SIG_SETMASK The resulting set is the signal set pointed to by the argument *set*.

12221 The **<signal.h>** header shall also define the following symbolic constants:

12222 CX SA_NOCLDSTOP Do not generate SIGCHLD when children stop
12223 XSI or stopped children continue.

12224 XSI SA_ONSTACK Causes signal delivery to occur on an alternate stack.

12225 CX SA_RESETHAND Causes signal dispositions to be set to SIG_DFL on entry to signal
12226 handlers.

12227 CX SA_RESTART Causes certain functions to become restartable.

12228 CX SA_SIGINFO Causes extra information to be passed to signal handlers at the time of
12229 receipt of a signal.

12230 XSI SA_NOCLDWAIT Causes implementations not to create zombie processes or status
12231 information on child termination. See *sigaction*( ).

12232 CX SA_NODEFER Causes signal not to be automatically blocked on entry to signal handler.

12233 XSI SS_ONSTACK Process is executing on an alternate signal stack.

12234 XSI SS_DISABLE Alternate signal stack is disabled.

12235 XSI MINSIGSTKSZ Minimum stack size for a signal handler.

12236 XSI SIGSTKSZ Default size in bytes for the alternate signal stack.

12237 CX The **<signal.h>** header shall define the **mcontext_t** type through **typedef**.

12238 CX The **<signal.h>** header shall define the **ucontext_t** type as a structure that shall include at least
12239 the following members:

```
12240 ucontext_t *uc_link      Pointer to the context that is resumed
12241                           when this context returns.
12242 sigset_t    uc_sigmask   The set of signals that are blocked when this
12243                           context is active.
12244 stack_t     uc_stack     The stack used by this context.
12245 mcontext_t  uc_mcontext  A machine-specific representation of the saved
12246                           context.
```

12247 The **<signal.h>** header shall define the **stack_t** type as a structure, which shall include at least
12248 the following members:

```
12249 void     *ss_sp      Stack base or pointer.
12250 size_t    ss_size    Stack size.
12251 int       ss_flags   Flags.
```


12252 CX The **<signal.h>** header shall define the **siginfo_t** type as a structure, which shall include at least
12253 the following members:

12254 CX	`int`	`si_signo`	Signal number.
12255	`int`	`si_code`	Signal code.
12256 XSI	`int`	`si_errno`	If non-zero, an *errno* value associated with
12257			this signal, as described in **<errno.h>**.
12258 CX	`pid_t`	`si_pid`	Sending process ID.
12259	`uid_t`	`si_uid`	Real user ID of sending process.
12260	`void`	`*si_addr`	Address that caused fault.
12261	`int`	`si_status`	Exit value or signal.
12262	`union sigval`	`si_value`	Signal value.

12263 CX The **<signal.h>** header shall define the symbolic constants in the **Code** column of the following
12264 table for use as values of *si_code* that are signal-specific or non-signal-specific reasons why the
12265 signal was generated.

	Signal	Code	Reason
CX	SIGILL	ILL_ILLOPC	Illegal opcode.
		ILL_ILLOPN	Illegal operand.
		ILL_ILLADR	Illegal addressing mode.
		ILL_ILLTRP	Illegal trap.
		ILL_PRVOPC	Privileged opcode.
		ILL_PRVREG	Privileged register.
		ILL_COPROC	Coprocessor error.
		ILL_BADSTK	Internal stack error.
	SIGFPE	FPE_INTDIV	Integer divide by zero.
		FPE_INTOVF	Integer overflow.
		FPE_FLTDIV	Floating-point divide by zero.
		FPE_FLTOVF	Floating-point overflow.
		FPE_FLTUND	Floating-point underflow.
		FPE_FLTRES	Floating-point inexact result.
		FPE_FLTINV	Invalid floating-point operation.
		FPE_FLTSUB	Subscript out of range.
	SIGSEGV	SEGV_MAPERR	Address not mapped to object.
		SEGV_ACCERR	Invalid permissions for mapped object.
	SIGBUS	BUS_ADRALN	Invalid address alignment.
		BUS_ADRERR	Nonexistent physical address.
		BUS_OBJERR	Object-specific hardware error.
XSI	SIGTRAP	TRAP_BRKPT	Process breakpoint.
		TRAP_TRACE	Process trace trap.
CX	SIGCHLD	CLD_EXITED	Child has exited.
		CLD_KILLED	Child has terminated abnormally with no additional actions.
		CLD_DUMPED	Child has terminated abnormally and additional actions may have been taken.
		CLD_TRAPPED	Traced child has trapped.
		CLD_STOPPED	Child has stopped.
		CLD_CONTINUED	Stopped child has continued.
	Any	SI_USER	Signal sent by *kill*().
		SI_QUEUE	Signal sent by *sigqueue*().
		SI_TIMER	Signal generated by expiration of a timer set by *timer_settime*().
		SI_ASYNCIO	Signal generated by completion of an asynchronous I/O request.
		SI_MESGQ	Signal generated by arrival of a message on an empty message queue

CX Implementations may support additional *si_code* values not included in this list, may generate values included in this list under circumstances other than those described in this list, and may contain extensions or limitations that prevent some values from being generated. Implementations do not generate a different value from the ones described in this list for circumstances described in this list.

12309 CX In addition, the following signal-specific information shall be available:

Signal	Member	Value
SIGILL SIGFPE	**void** * *si_addr*	Address of faulting instruction.
SIGSEGV SIGBUS	**void** * *si_addr*	Address of faulting memory reference.
SIGCHLD	**pid_t** *si_pid* **int** *si_status*	Child process ID. If *si_code* is equal to CLD_EXITED, then *si_status* holds the exit value of the process; otherwise, it is equal to the signal that caused the process to change state. The exit value in *si_status* shall be equal to the full exit value (that is, the value passed to *_exit*(), *_Exit*(), or *exit*(), or returned from *main*()); it shall not be limited to the least significant eight bits of the value.
	uid_t *si_uid*	Real user ID of the process that sent the signal.

12323 For some implementations, the value of *si_addr* may be inaccurate.

12324 The following shall be declared as functions and may also be defined as macros. Function
12325 prototypes shall be provided.

```
12326 CX      int    kill(pid_t, int);
12327 XSI     int    killpg(pid_t, int);
12328 CX      void   psiginfo(const siginfo_t *, const char *);
12329         void   psignal(int, const char *);
12330         int    pthread_kill(pthread_t, int);
12331         int    pthread_sigmask(int, const sigset_t *restrict,
12332                    sigset_t *restrict);
12333         int    raise(int);
12334 CX      int    sig2str(int, char *);
12335         int    sigaction(int, const struct sigaction *restrict,
12336                    struct sigaction *restrict);
12337         int    sigaddset(sigset_t *, int);
12338 XSI     int    sigaltstack(const stack_t *restrict, stack_t *restrict);
12339 CX      int    sigdelset(sigset_t *, int);
12340         int    sigemptyset(sigset_t *);
12341         int    sigfillset(sigset_t *);
12342         int    sigismember(const sigset_t *, int);
12343         void (*signal(int, void (*)(int)))(int);
12344 CX      int    sigpending(sigset_t *);
12345         int    sigprocmask(int, const sigset_t *restrict, sigset_t *restrict);
12346         int    sigqueue(pid_t, int, union sigval);
12347         int    sigsuspend(const sigset_t *);
12348         int    sigtimedwait(const sigset_t *restrict, siginfo_t *restrict,
12349                    const struct timespec *restrict);
12350         int    sigwait(const sigset_t *restrict, int *restrict);
12351         int    sigwaitinfo(const sigset_t *restrict, siginfo_t *restrict);
12352         int    str2sig(const char *restrict, int *restrict);
```

12353 CX Inclusion of the **<signal.h>** header may make visible all symbols from the **<time.h>** header.

12354 **APPLICATION USAGE**

12355 On systems not supporting the XSI option, the *si_pid* and *si_uid* members of **siginfo_t** are only
12356 required to be valid when *si_code* is SI_USER or SI_QUEUE. On XSI-conforming systems, they
12357 are also valid for all *si_code* values less than or equal to 0; however, it is unspecified whether
12358 SI_USER and SI_QUEUE have values less than or equal to zero, and therefore XSI applications
12359 should check whether *si_code* has the value SI_USER or SI_QUEUE or is less than or equal to 0 to
12360 tell whether *si_pid* and *si_uid* are valid.

12361 **RATIONALE**

12362 None.

12363 **FUTURE DIRECTIONS**

12364 None.

12365 **SEE ALSO**

12366 **<errno.h>**, **<limits.h>**, **<sys/types.h>**, **<time.h>**

12367 XSH Section 2.2 (on page 496), *alarm*( ), *kill*( ), *killpg*( ), *psiginfo*( ), *pthread_kill*( ), *pthread_sigmask*( ),
12368 *raise*( ), *sig2str*( ), *sigaction*( ), *sigaddset*( ), *sigaltstack*( ), *sigdelset*( ), *sigemptyset*( ), *sigfillset*( ),
12369 *sigismember*( ), *signal*( ), *sigpending*( ), *sigqueue*( ), *sigsuspend*( ), *sigtimedwait*( ), *sigwait*( ),
12370 *timer_create*( ), *wait*( ), *waitid*( )

12371 XCU *kill*

12372 **CHANGE HISTORY**

12373 First released in Issue 1.

12374 **Issue 5**

12375 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
12376 Threads Extension.

12377 The default action for SIGURG is changed from i to iii. The function prototype for *sigmask*( ) is
12378 removed.

12379 **Issue 6**

12380 The Open Group Corrigendum U035/2 is applied. In the DESCRIPTION, the wording for
12381 abnormal termination is clarified.

12382 The Open Group Corrigendum U028/8 is applied, correcting the prototype for the *sigset*( )
12383 function.

12384 The Open Group Corrigendum U026/3 is applied, correcting the type of the *sigev_notify_function*
12385 function member of the **sigevent** structure.

12386 The following new requirements on POSIX implementations derive from alignment with the
12387 Single UNIX Specification:

12388 • The SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU signals are now
12389 mandated. This is also a FIPS requirement.

12390 • The **pid_t** definition is mandated.

12391 The RT markings are changed to RTS to denote that the semantics are part of the Realtime
12392 Signals Extension option.

12393 The **restrict** keyword is added to the prototypes for *sigaction*( ), *sigaltstack*( ), *sigprocmask*( ),
12394 *sigtimedwait*( ), *sigwait*( ), and *sigwaitinfo*( ).

12395 IEEE PASC Interpretation 1003.1 #85 is applied, adding the statement that symbols from
12396 **<time.h>** may be made visible when **<signal.h>** is included.

Extensions beyond the ISO C standard are marked.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/14 is applied, changing the descriptive text for members of the **sigaction** structure.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/15 is applied, correcting the definition of the *sa_sigaction* member of the **sigaction** structure.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/24 is applied, reworking the ordering of the **siginfo_t** type structure in the DESCRIPTION. This is an editorial change and no normative change is intended.

Issue 7

SD5-XBD-ERN-5 is applied.

SD5-XBD-ERN-39 is applied, removing the **sigstack** structure which should have been removed at the same time as the LEGACY *sigstack*( ) function.

SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

Austin Group Interpretation 1003.1-2001 #034 is applied.

The **ucontext_t** and **mcontext_t** structures are added here from the obsolescent **<ucontext.h>** header.

The *psiginfo*( ) and *psignal*( ) functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The SIGPOLL and SIGPROF signals and text relating to the XSI STREAMS option are marked obsolescent.

The SA_RESETHAND, SA_RESTART, SA_SIGINFO, SA_NOCLDWAIT, and SA_NODEFER constants are moved from the XSI option to the Base.

Functionality relating to the Realtime Signals Extension option is moved to the Base.

This reference page is clarified with respect to macros and symbolic constants, and declarations for the **pthread_attr_t**, **pthread_t**, and **uid_t** types and the **timespec** structure are added.

SIGRTMIN and SIGRTMAX are required to be positive integer expressions.

The APPLICATION USAGE section is updated to describe the *si_pid* and *si_uid* members of **siginfo_t**.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0062 [208], XBD/TC1-2008/0063 [80], and XBD/TC1-2008/0064 [157] are applied.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0070 [536], XBD/TC2-2008/0071 [690], XBD/TC2-2008/0072 [594], XBD/TC2-2008/0073 [844], and XBD/TC2-2008/0074 [536] are applied.

Issue 8

Austin Group Defect 741 is applied, restricting the value of SIGRTMAX to less than the value returned by *sysconf*(_SC_NSIG) and the value of macros that are used to refer to the signals to less than {NSIG_MAX}.

Austin Group Defect 1138 is applied, adding *sig2str*( ) and *str2sig*( ).

Austin Group Defect 1141 is applied, changing the descriptions of CLD_KILLED and CLD_DUMPED.

Austin Group Defect 1151 is applied, adding SIGWINCH.

Austin Group Defect 1215 is applied, removing XSI shading from text relating to abnormal process termination with additional actions.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1775 is applied, changing the description of the *si_addr* member of **siginfo_t**.

NAME

spawn.h — spawn (**ADVANCED REALTIME**)

SYNOPSIS

SPN
```
#include <spawn.h>
```

DESCRIPTION

The **<spawn.h>** header shall define the **posix_spawnattr_t** and **posix_spawn_file_actions_t** types used in performing spawn operations.

The **<spawn.h>** header shall define the **mode_t** and **pid_t** types as described in **<sys/types.h>**.

The **<spawn.h>** header shall define the **sigset_t** type as described in **<signal.h>**.

The tag **sched_param** shall be declared as naming an incomplete structure type, the contents of which are described in the **<sched.h>** header.

The **<spawn.h>** header shall define the following symbolic constants for use as the flags that may be set in a **posix_spawnattr_t** object using the *posix_spawnattr_setflags*( ) function:

POSIX_SPAWN_RESETIDS
POSIX_SPAWN_SETPGROUP
PS POSIX_SPAWN_SETSCHEDPARAM
POSIX_SPAWN_SETSCHEDULER
POSIX_SPAWN_SETSID
POSIX_SPAWN_SETSIGDEF
POSIX_SPAWN_SETSIGMASK

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int   posix_spawn(pid_t *restrict, const char *restrict,
          const posix_spawn_file_actions_t *restrict,
          const posix_spawnattr_t *restrict, char *const [restrict],
          char *const [restrict]);
int   posix_spawn_file_actions_addchdir(posix_spawn_file_actions_t
          *restrict, const char *restrict);
int   posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *,
          int);
int   posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *,
          int, int);
int   posix_spawn_file_actions_addfchdir(posix_spawn_file_actions_t *,
          int);
int   posix_spawn_file_actions_addopen(posix_spawn_file_actions_t
          *restrict, int, const char *restrict, int, mode_t);
int   posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *);
int   posix_spawn_file_actions_init(posix_spawn_file_actions_t *);
int   posix_spawnattr_destroy(posix_spawnattr_t *);
int   posix_spawnattr_getflags(const posix_spawnattr_t *restrict,
          short *restrict);
int   posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict,
          pid_t *restrict);
PS int   posix_spawnattr_getschedparam(const posix_spawnattr_t *restrict,
          struct sched_param *restrict);
int   posix_spawnattr_getschedpolicy(const posix_spawnattr_t *restrict,
          int *restrict);
```



```
12490          int   posix_spawnattr_getsigdefault(const posix_spawnattr_t *restrict,
12491                    sigset_t *restrict);
12492          int   posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict,
12493                    sigset_t *restrict);
12494          int   posix_spawnattr_init(posix_spawnattr_t *);
12495          int   posix_spawnattr_setflags(posix_spawnattr_t *, short);
12496          int   posix_spawnattr_setpgroup(posix_spawnattr_t *, pid_t);
12497    PS    int   posix_spawnattr_setschedparam(posix_spawnattr_t *restrict,
12498                    const struct sched_param *restrict);
12499          int   posix_spawnattr_setschedpolicy(posix_spawnattr_t *, int);
12500          int   posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict,
12501                    const sigset_t *restrict);
12502          int   posix_spawnattr_setsigmask(posix_spawnattr_t *restrict,
12503                    const sigset_t *restrict);
12504          int   posix_spawnp(pid_t *restrict, const char *restrict,
12505                    const posix_spawn_file_actions_t *restrict,
12506                    const posix_spawnattr_t *restrict,
12507                    char *const [restrict], char *const [restrict]);
```

12508 Inclusion of the **<spawn.h>** header may make visible symbols defined in the **<sched.h>** and
12509 **<signal.h>** headers.

12510 **APPLICATION USAGE**

12511 None.

12512 **RATIONALE**

12513 None.

12514 **FUTURE DIRECTIONS**

12515 None.

12516 **SEE ALSO**

12517 **<sched.h>**, **<semaphore.h>**, **<signal.h>**, **<sys/types.h>**

12518 XSH *posix_spawn*( ), *posix_spawn_file_actions_addclose*( ), *posix_spawn_file_actions_adddup2*( ),
12519 *posix_spawn_file_actions_destroy*( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_getflags*( ),
12520 *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ),
12521 *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getsigmask*( )

12522 **CHANGE HISTORY**

12523 First released in Issue 6. Included for alignment with IEEE Std 1003.1d-1999.

12524 The **restrict** keyword is added to the prototypes for *posix_spawn*( ),
12525 *posix_spawn_file_actions_addopen*( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ),
12526 *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ),
12527 *posix_spawnattr_getsigmask*( ), *posix_spawnattr_setsigdefault*( ), *posix_spawnattr_setschedparam*( ),
12528 *posix_spawnattr_setsigmask*( ), and *posix_spawnp*( ).

12529 **Issue 7**

12530 This reference page is clarified with respect to macros and symbolic constants, and declarations
12531 for the **mode_t**, **pid_t**, and **sigset_t** types are added.

12532 **Issue 8**

12533 Austin Group Defect 1044 is applied, adding POSIX_SPAWN_SETSID.

12534 Austin Group Defect 1208 is applied, adding *posix_spawn_file_actions_addchdir*( ) and
12535 *posix_spawn_file_actions_addfchdir*( ).

12536 Austin Group Defect 1328 is applied, adding the **restrict** keyword to the third parameter of
12537 *posix_spawn*( ) and *posix_spawnp*( ).

NAME

stdalign.h — alignment macros

SYNOPSIS

```
#include <stdalign.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The **<stdalign.h>** header shall define the following macros:

alignas Expands to **_Alignas**

alignof Expands to **_Alignof**

__alignas_is_defined
Expands to the integer constant 1

__alignof_is_defined
Expands to the integer constant 1

The __alignas_is_defined and __alignof_is_defined macros shall be suitable for use in **#if** preprocessing directives.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

None.

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

12565 **NAME**

12566 stdatomic.h — atomics

12567 **SYNOPSIS**

12568 `#include <stdatomic.h>`

12569 **DESCRIPTION**

12570 CX The functionality described on this reference page is aligned with the ISO C standard. Any
12571 conflict between the requirements described here and the ISO C standard is unintentional. This
12572 volume of POSIX.1-2024 defers to the ISO C standard.

12573 Implementations that define the macro __STDC_NO_ATOMICS__ need not provide this header
12574 nor support any of its facilities.

12575 The **<stdatomic.h>** header shall define the **atomic_flag** type as a structure type. This type
12576 provides the classic test-and-set functionality. It shall have two states, set and clear. Operations
12577 on an object of type **atomic_flag** shall be lock free.

12578 The **<stdatomic.h>** header shall define each of the atomic integer types in the following table as
12579 a type that has the same representation and alignment requirements as the corresponding direct
12580 type.

12581 **Note:** The same representation and alignment requirements are meant to imply interchangeability as
12582 arguments to functions, return values from functions, and members of unions.

Atomic type name	Direct type
atomic_bool	**_Atomic _Bool**
atomic_char	**_Atomic char**
atomic_schar	**_Atomic signed char**
atomic_uchar	**_Atomic unsigned char**
atomic_short	**_Atomic short**
atomic_ushort	**_Atomic unsigned short**
atomic_int	**_Atomic int**
atomic_uint	**_Atomic unsigned int**
atomic_long	**_Atomic long**
atomic_ulong	**_Atomic unsigned long**
atomic_llong	**_Atomic long long**
atomic_ullong	**_Atomic unsigned long long**
atomic_char16_t	**_Atomic char16_t**
atomic_char32_t	**_Atomic char32_t**
atomic_wchar_t	**_Atomic wchar_t**
atomic_int_least8_t	**_Atomic int_least8_t**
atomic_uint_least8_t	**_Atomic uint_least8_t**
atomic_int_least16_t	**_Atomic int_least16_t**
atomic_uint_least16_t	**_Atomic uint_least16_t**
atomic_int_least32_t	**_Atomic int_least32_t**
atomic_uint_least32_t	**_Atomic uint_least32_t**
atomic_int_least64_t	**_Atomic int_least64_t**
atomic_uint_least64_t	**_Atomic uint_least64_t**
atomic_int_fast8_t	**_Atomic int_fast8_t**
atomic_uint_fast8_t	**_Atomic uint_fast8_t**
atomic_int_fast16_t	**_Atomic int_fast16_t**
atomic_uint_fast16_t	**_Atomic uint_fast16_t**
atomic_int_fast32_t	**_Atomic int_fast32_t**
atomic_uint_fast32_t	**_Atomic uint_fast32_t**
atomic_int_fast64_t	**_Atomic int_fast64_t**
atomic_uint_fast64_t	**_Atomic uint_fast64_t**
atomic_intptr_t	**_Atomic intptr_t**
atomic_uintptr_t	**_Atomic uintptr_t**
atomic_size_t	**_Atomic size_t**
atomic_ptrdiff_t	**_Atomic ptrdiff_t**
atomic_intmax_t	**_Atomic intmax_t**
atomic_uintmax_t	**_Atomic uintmax_t**

The **<stdatomic.h>** header shall define the **memory_order** type as an enumerated type whose enumerators shall include at least the following:

```
memory_order_relaxed
memory_order_consume
memory_order_acquire
memory_order_release
memory_order_acq_rel
memory_order_seq_cst
```

The **<stdatomic.h>** header shall define the following atomic lock-free macros:

ATOMIC_BOOL_LOCK_FREE
ATOMIC_CHAR_LOCK_FREE
ATOMIC_CHAR16_T_LOCK_FREE

ATOMIC_CHAR32_T_LOCK_FREE
ATOMIC_WCHAR_T_LOCK_FREE
ATOMIC_SHORT_LOCK_FREE
ATOMIC_INT_LOCK_FREE
ATOMIC_LONG_LOCK_FREE
ATOMIC_LLONG_LOCK_FREE
ATOMIC_POINTER_LOCK_FREE

which shall expand to constant expressions suitable for use in **#if** preprocessing directives and which shall indicate the lock-free property of the corresponding atomic types (both signed and unsigned). A value of 0 shall indicate that the type is never lock-free; a value of 1 shall indicate that the type is sometimes lock-free; a value of 2 shall indicate that the type is always lock-free.

The **<stdatomic.h>** header shall define the macro ATOMIC_FLAG_INIT which shall expand to an initializer for an object of type **atomic_flag**. This macro shall initialize an **atomic_flag** to the clear state. An **atomic_flag** that is not explicitly initialized with ATOMIC_FLAG_INIT is initially in an indeterminate state.

OB The **<stdatomic.h>** header shall define the macro ATOMIC_VAR_INIT(*value*) which shall expand to a token sequence suitable for initializing an atomic object of a type that is initialization-compatible with the non-atomic type of its *value* argument. An atomic object with automatic storage duration that is not explicitly initialized is initially in an indeterminate state.

The **<stdatomic.h>** header shall define the macro *kill_dependency*() which shall behave as described in *kill_dependency*().

The **<stdatomic.h>** header shall declare the following generic functions, where ***A*** refers to an atomic type, ***C*** refers to its corresponding non-atomic type, and ***M*** is ***C*** for atomic integer types or **ptrdiff_t** for atomic pointer types.

```
_Bool   atomic_compare_exchange_strong(volatile A *, C *, C);
_Bool   atomic_compare_exchange_strong_explicit(volatile A *,
                C *, C, memory_order, memory_order);
_Bool   atomic_compare_exchange_weak(volatile A *, C *, C);
_Bool   atomic_compare_exchange_weak_explicit(volatile A *, C *,
                C, memory_order, memory_order);
C       atomic_exchange(volatile A *, C);
C       atomic_exchange_explicit(volatile A *, C, memory_order);
C       atomic_fetch_add(volatile A *, M);
C       atomic_fetch_add_explicit(volatile A *, M,
                memory_order);
C       atomic_fetch_and(volatile A *, M);
C       atomic_fetch_and_explicit(volatile A *, M,
                memory_order);
C       atomic_fetch_or(volatile A *, M);
C       atomic_fetch_or_explicit(volatile A *, M, memory_order);
C       atomic_fetch_sub(volatile A *, M);
C       atomic_fetch_sub_explicit(volatile A *, M,
                memory_order);
C       atomic_fetch_xor(volatile A *, M);
C       atomic_fetch_xor_explicit(volatile A *, M,
                memory_order);
void    atomic_init(volatile A *, C);
_Bool   atomic_is_lock_free(const volatile A *);
C       atomic_load(const volatile A *);
```



```
C       atomic_load_explicit(const volatile A *, memory_order);
void    atomic_store(volatile A *, C);
void    atomic_store_explicit(volatile A *, C, memory_order);
```

It is unspecified whether any generic function declared in **<stdatomic.h>** is a macro or an identifier declared with external linkage. If a macro definition is suppressed in order to access an actual function, or a program defines an external identifier with the name of a generic function, the behavior is undefined.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
void    atomic_flag_clear(volatile atomic_flag *);
void    atomic_flag_clear_explicit(volatile atomic_flag *,
                memory_order);
_Bool   atomic_flag_test_and_set(volatile atomic_flag *);
_Bool   atomic_flag_test_and_set_explicit(
                volatile atomic_flag *, memory_order);
void    atomic_signal_fence(memory_order);
void    atomic_thread_fence(memory_order);
```

APPLICATION USAGE

None.

RATIONALE

Since operations on the **atomic_flag** type are lock free, the operations should also be address-free. No other type requires lock-free operations, so the **atomic_flag** type is the minimum hardware-implemented type needed to conform to this standard. The remaining types can be emulated with **atomic_flag**, though with less than ideal properties.

The representation of atomic integer types need not have the same size as their corresponding regular types. They should have the same size whenever possible, as it eases effort required to port existing code.

FUTURE DIRECTIONS

The ISO C standard states that the macro ATOMIC_VAR_INIT is an obsolescent feature. This macro may be removed in a future version of this standard.

SEE ALSO

Section 4.15.1

XSH *atomic_compare_exchange_strong*( ), *atomic_exchange*( ), *atomic_fetch_add*( ), *atomic_flag_clear*( ), *atomic_flag_test_and_set*( ), *atomic_init*( ), *atomic_is_lock_free*( ), *atomic_load*( ), *atomic_signal_fence*( ), *atomic_store*( ), *kill_dependency*( ).

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

12719 **NAME**

12720 stdarg.h — handle variable argument list

12721 **SYNOPSIS**

12722 ```
#include <stdarg.h>

void va_start(va_list ap, argN);
void va_copy(va_list dest, va_list src);
type va_arg(va_list ap, type);
void va_end(va_list ap);
```

12727 **DESCRIPTION**

12728 CX The functionality described on this reference page is aligned with the ISO C standard. Any
12729 conflict between the requirements described here and the ISO C standard is unintentional. This
12730 volume of POSIX.1-2024 defers to the ISO C standard.

12731 The **<stdarg.h>** header shall contain a set of macros which allows portable functions that accept
12732 variable argument lists to be written. Functions that have variable argument lists (such as
12733 *printf*()) but do not use these macros are inherently non-portable, as different systems use
12734 different argument-passing conventions.

12735 The **<stdarg.h>** header shall define the **va_list** type for variables used to traverse the list.

12736 The *va_start*() macro is invoked to initialize *ap* to the beginning of the list before any calls to
12737 *va_arg*().

12738 The *va_copy*() macro initializes *dest* as a copy of *src*, as if the *va_start*() macro had been applied
12739 to *dest* followed by the same sequence of uses of the *va_arg*() macro as had previously been used
12740 to reach the present state of *src*. Neither the *va_copy*() nor *va_start*() macro shall be invoked to
12741 reinitialize *dest* without an intervening invocation of the *va_end*() macro for the same *dest*.

12742 The object *ap* may be passed as an argument to another function; if that function invokes the
12743 *va_arg*() macro with parameter *ap*, the value of *ap* in the calling function is unspecified and shall
12744 be passed to the *va_end*() macro prior to any further reference to *ap*. The parameter *argN* is the
12745 identifier of the rightmost parameter in the variable parameter list in the function definition (the
12746 one just before the ...). If the parameter *argN* is declared with the **register** storage class, with a
12747 function type or array type, or with a type that is not compatible with the type that results after
12748 application of the default argument promotions, the behavior is undefined.

12749 The *va_arg*() macro shall return the next argument in the list pointed to by *ap*. Each invocation
12750 of *va_arg*() modifies *ap* so that the values of successive arguments are returned in turn. The *type*
12751 parameter shall be a type name specified such that the type of a pointer to an object that has the
12752 specified type can be obtained simply by postfixing a `'*'` to type. If there is no actual next
12753 argument, or if *type* is not compatible with the type of the actual next argument (as promoted
12754 according to the default argument promotions), the behavior is undefined, except for the
12755 following cases:

12756 • One type is a signed integer type, the other type is the corresponding unsigned integer
12757 type, and the value is representable in both types.

12758 • One type is a pointer to **void** and the other is a pointer to a character type.

12759 XSI • Both types are pointers.

12760 Different types can be mixed, but it is up to the routine to know what type of argument is
12761 expected.

12762 The *va_end*() macro is used to clean up; it invalidates *ap* for use (unless *va_start*() or *va_copy*() is
12763 invoked again).

  
```

Each invocation of the *va_start*( ) and *va_copy*( ) macros shall be matched by a corresponding invocation of the *va_end*( ) macro in the same function.

Multiple traversals, each bracketed by *va_start*( ) ... *va_end*( ), are possible.

EXAMPLES

This example is a possible implementation of *execl*( ):

```
#include <stdarg.h>

#define  MAXARGS     31

/*
 * execl is called by
 * execl(file, arg1, arg2, ..., (char *)(0));
 */
int execl(const char *file, const char *args, ...)
{
    va_list ap;
    char *array[MAXARGS +1];
    int argno = 0;

    va_start(ap, args);
    while (args != 0 && argno < MAXARGS)
    {
        array[argno++] = args;
        args = va_arg(ap, const char *);
    }
    array[argno] = (char *) 0;
    va_end(ap);
    return execv(file, array);
}
```

APPLICATION USAGE

It is up to the calling routine to communicate to the called routine how many arguments there are, since it is not always possible for the called routine to determine this in any other way. For example, *execl*( ) is passed a null pointer to signal the end of the list. The *printf*( ) function can tell how many arguments are there by the *format* argument.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH *exec*, *fprintf*( )

CHANGE HISTORY

First released in Issue 4. Derived from the ANSI C standard.

Issue 6

This reference page is updated to align with the ISO/IEC 9899: 1999 standard.

12805 **NAME**

12806 stdbool.h — boolean type and values

12807 **SYNOPSIS**

12808 `#include <stdbool.h>`

12809 **DESCRIPTION**

12810 CX The functionality described on this reference page is aligned with the ISO C standard. Any
12811 conflict between the requirements described here and the ISO C standard is unintentional. This
12812 volume of POSIX.1-2024 defers to the ISO C standard.

12813 The **<stdbool.h>** header shall define the following macros:

12814 bool Expands to **_Bool**.

12815 true Expands to the integer constant 1.

12816 false Expands to the integer constant 0.

12817 __bool_true_false_are_defined
12818 Expands to the integer constant 1.

12819 The macros true, false and __bool_true_false_are_defined shall be suitable for use in **#if**
12820 preprocessing directives.

12821 OB An application can undefine and then possibly redefine the macros bool, true, and false.

12822 **APPLICATION USAGE**

12823 None.

12824 **RATIONALE**

12825 None.

12826 **FUTURE DIRECTIONS**

12827 The ability to undefine and redefine the macros bool, true, and false is an obsolescent feature
12828 and may be removed in a future version.

12829 **SEE ALSO**

12830 None.

12831 **CHANGE HISTORY**

12832 First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

12833 **Issue 8**

12834 Austin Group Defect 1257 is applied, correcting a mismatch with the ISO C standard by adding
12835 the requirement for the integer-valued macros to be suitable for use in **#if** preprocessing
12836 directives.

12837 Austin Group Defect 1302 is applied, adding OB shading to the statement that an application
12838 can ``undefine and then possibly redefine the macros bool, true, and false''.

NAME

stddef.h — standard type definitions

SYNOPSIS

```
#include <stddef.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The **<stddef.h>** header shall define the following macros:

CX NULL Null pointer constant. The macro shall expand to an integer constant expression with the value 0 cast to type **void** *. Additionally, any pointer object whose representation has all bits set to zero, perhaps by *memset*( ) to 0 or by *calloc*( ), shall be treated as a null pointer.

offsetof(*type*, *member-designator*)
Integer constant expression of type **size_t**, the value of which is the offset in bytes to the structure member (*member-designator*), from the beginning of its structure (*type*).

The **<stddef.h>** header shall define the following types:

max_align_t Object type whose alignment is the greatest fundamental alignment.

ptrdiff_t Signed integer type of the result of subtracting two pointers.

wchar_t Integer type whose range of values can represent distinct codes for all members of the largest extended character set specified among the supported locales; the null character shall have the code value zero. Each member of the basic character set shall have a code value equal to its value when used as the lone character in an integer character constant if an implementation does not define __STDC_MB_MIGHT_NEQ_WC__.

size_t Unsigned integer type of the result of the *sizeof* operator.

The implementation shall support one or more programming environments in which the widths of **ptrdiff_t**, **size_t**, and **wchar_t** are no greater than the width of type **long**. The names of these programming environments can be obtained using the *confstr*( ) function or the *getconf* utility.

APPLICATION USAGE

None.

RATIONALE

The ISO C standard does not require the NULL macro to include the cast to type **void** * and specifies that the NULL macro be implementation-defined. POSIX.1-2024 requires the cast and therefore need not be implementation-defined.

Likewise, the ISO C standard does not require a pointer object whose representation has all bits set to zero to be treated as a null pointer. While there has been historical hardware where non-zero patterns were more efficient for use as the canonical null pointer, no known POSIX system has tried to target such hardware. However, though unlikely in modern hardware, a compiler is still allowed to treat more than one bit pattern as a representation of the null pointer (all such patterns will compare equal to one another, and unequal to any pointer to any other object). Thus, applications should not assume that a pointer object with non-zero representation is not a null pointer.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, **<wchar.h>**

XSH *confstr( )*

XCU *getconf*

CHANGE HISTORY

First released in Issue 4. Derived from the ANSI C standard.

Issue 7

This reference page is clarified with respect to macros and symbolic constants.

SD5-XBD-ERN-53 is applied, updating the definition of **wchar_t** to align with ISO/IEC 9899:1999 standard, Technical Corrigendum 3.

Issue 8

Austin Group Defect 940 is applied, adding a requirement that any pointer object whose representation has all bits set to zero is interpreted as a null pointer.

Austin Group Defect 1302 is applied, adding **max_align_t**.

NAME

stdint.h — integer types

SYNOPSIS

```
#include <stdint.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to enable the visibility of these symbols in this header.

The **<stdint.h>** header shall declare sets of integer types having specified widths, and shall define corresponding sets of macros. It shall also define macros that specify limits of integer types corresponding to types defined in other standard headers.

Note: The ``width'' of an integer type is the number of bits used to store its value in a pure binary system; the actual type may use more bits than that (for example, a 28-bit type could be stored in 32 bits of actual storage). An *N*-bit signed type in two's complement representation has values in the range -2^{N-1} to $2^{N-1}-1$, while an *N*-bit unsigned type has values in the range 0 to 2^N-1. While the ISO C standard also permits signed integers in sign-magnitude or one's complement form, this standard requires an implementation to use two's complement representation for the standard integer types.

Types are defined in the following categories:

- Integer types having certain exact widths
- Integer types having at least certain specified widths
- Fastest integer types having at least certain specified widths
- Integer types wide enough to hold pointers to objects
- Integer types having greatest width

(Some of these types may denote the same type.)

Corresponding macros specify limits of the declared types and construct suitable constants.

For each type described herein that the implementation provides, the **<stdint.h>** header shall declare that **typedef** name and define the associated macros. Conversely, for each type described herein that the implementation does not provide, the **<stdint.h>** header shall not declare that **typedef** name, nor shall it define the associated macros. An implementation shall provide those types described as required, but need not provide any of the others (described as optional).

Integer Types

When **typedef** names differing only in the absence or presence of the initial *u* are defined, they shall denote corresponding signed and unsigned types as described in the ISO C standard, Section 6.2.5; an implementation providing one of these corresponding types shall also provide the other.

In the following descriptions, the symbol *N* represents an unsigned decimal integer with no leading zeros (for example, 8 or 24, but not 04 or 048).

- Exact-width integer types

  The **typedef** name **int***N***_t** designates a signed integer type with width *N*, no padding bits, and a two's-complement representation. Thus, **int8_t** denotes such a signed integer type with a width of exactly 8 bits.

  The **typedef** name **uint***N***_t** designates an unsigned integer type with width *N* and no

padding bits. Thus, **uint24_t** denotes such an unsigned integer type with a width of exactly 24 bits.

CX The following types are required:

int8_t
int16_t
int32_t
uint8_t
uint16_t
uint32_t

If an implementation provides integer types with width 64 that meet these requirements, then the following types are required:

int64_t
uint64_t

CX In particular, this is the case if any of the following are true:

— The implementation supports the _POSIX_V8_ILP32_OFFBIG programming environment and the application is being built in the _POSIX_V8_ILP32_OFFBIG programming environment (see the Shell and Utilities volume of POSIX.1-2024, *c17*, Programming Environments).

— The implementation supports the _POSIX_V8_LP64_OFF64 programming environment and the application is being built in the _POSIX_V8_LP64_OFF64 programming environment.

— The implementation supports the _POSIX_V8_LPBIG_OFFBIG programming environment and the application is being built in the _POSIX_V8_LPBIG_OFFBIG programming environment.

If the representation of any of the standard types **short**, **int**, **long** or **long long** is not the same as one of the above required types, an **int***N***_t** type with that representation shall be defined along with its **uint***N***_t** counterpart.

All other types of this form are optional.

- Minimum-width integer types

The **typedef** name **int_least***N***_t** designates a signed integer type with a width of at least *N*, such that no signed integer type with lesser size has at least the specified width. Thus, **int_least32_t** denotes a signed integer type with a width of at least 32 bits.

The **typedef** name **uint_least***N***_t** designates an unsigned integer type with a width of at least *N*, such that no unsigned integer type with lesser size has at least the specified width. Thus, **uint_least16_t** denotes an unsigned integer type with a width of at least 16 bits.

The following types are required:

int_least8_t
int_least16_t
int_least32_t
int_least64_t
uint_least8_t
uint_least16_t
uint_least32_t
uint_least64_t

All other types of this form are optional.

- Fastest minimum-width integer types

Each of the following types designates an integer type that is usually fastest to operate with among all integer types that have at least the specified width.

The designated type is not guaranteed to be fastest for all purposes; if the implementation has no clear grounds for choosing one type over another, it may simply pick some integer type satisfying the signedness and width requirements.

The **typedef** name **int_fast***N***_t** designates the fastest signed integer type with a width of at least *N*. The **typedef** name **uint_fast***N***_t** designates the fastest unsigned integer type with a width of at least *N*.

The following types are required:

int_fast8_t
int_fast16_t
int_fast32_t
int_fast64_t
uint_fast8_t
uint_fast16_t
uint_fast32_t
uint_fast64_t

All other types of this form are optional.

- Integer types capable of holding object pointers

The following type designates a signed integer type with the property that any valid pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and the result shall compare equal to the original pointer:

intptr_t

The following type designates an unsigned integer type with the property that any valid pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and the result shall compare equal to the original pointer:

uintptr_t

XSI On XSI-conformant systems, the **intptr_t** and **uintptr_t** types are required; otherwise, they are optional.

- Greatest-width integer types

CX The following type designates a signed integer type using two's complement representation capable of representing any value of any signed integer type:

intmax_t

The following type designates an unsigned integer type capable of representing any value of any unsigned integer type:

uintmax_t

These types are required.

Note: Applications can test for optional types by using the corresponding limit macro from Limits of Specified-Width Integer Types.

Limits of Specified-Width Integer Types

The following macros specify the minimum and maximum limits of the types declared in the **<stdint.h>** header. Each macro name corresponds to a similar type name in Integer Types (on page 369).

Each instance of any defined macro shall be replaced by a constant expression suitable for use in **#if** preprocessing directives, and this expression shall have the same type as would an expression that is an object of the corresponding type converted according to the integer promotions. Its implementation-defined value shall be equal to or greater in magnitude (absolute value) than the corresponding value given below, with the same sign, except where stated to be exactly the given value.

- Limits of exact-width integer types
  - — Minimum values of exact-width signed integer types:

    {INT*N*_MIN} Exactly $-(2^{N-1})$
  - — Maximum values of exact-width signed integer types:

    {INT*N*_MAX} Exactly $2^{N-1}-1$
  - — Maximum values of exact-width unsigned integer types:

    {UINT*N*_MAX} Exactly $2^{N}-1$
- Limits of minimum-width integer types
  - — Minimum values of minimum-width signed integer types:

    CX {INT_LEAST*N*_MIN} $-(2^{N-1})$
  - — Maximum values of minimum-width signed integer types:

    {INT_LEAST*N*_MAX} $2^{N-1}-1$
  - — Maximum values of minimum-width unsigned integer types:

    {UINT_LEAST*N*_MAX} $2^{N}-1$
- Limits of fastest minimum-width integer types
  - — Minimum values of fastest minimum-width signed integer types:

    CX {INT_FAST*N*_MIN} $-(2^{N-1})$
  - — Maximum values of fastest minimum-width signed integer types:

13055 {INT_FAST*N*_MAX} $2^{N-1} - 1$

13056 — Maximum values of fastest minimum-width unsigned integer types:

13057 {UINT_FAST*N*_MAX} $2^{N} - 1$

13058 • Limits of integer types capable of holding object pointers

13059 — Minimum value of pointer-holding signed integer type:

13060 CX {INTPTR_MIN} $-(2^{15})$

13061 — Maximum value of pointer-holding signed integer type:

13062 {INTPTR_MAX} $2^{15} - 1$

13063 — Maximum value of pointer-holding unsigned integer type:

13064 {UINTPTR_MAX} $2^{16} - 1$

13065 • Limits of greatest-width integer types

13066 — Minimum value of greatest-width signed integer type:

13067 CX {INTMAX_MIN} $-(2^{63})$

13068 — Maximum value of greatest-width signed integer type:

13069 {INTMAX_MAX} $2^{63} - 1$

13070 — Maximum value of greatest-width unsigned integer type:

13071 {UINTMAX_MAX} $2^{64} - 1$

13072 **Limits of Other Integer Types**

13073 The following macros specify the minimum and maximum limits of integer types corresponding
13074 to types defined in other standard headers.

13075 Each instance of these macros shall be replaced by a constant expression suitable for use in **#if**
13076 preprocessing directives, and this expression shall have the same type as would an expression
13077 that is an object of the corresponding type converted according to the integer promotions. Its
13078 implementation-defined value shall be equal to or greater in magnitude (absolute value) than
13079 the corresponding value given below, with the same sign.

13080 • Limits of **ptrdiff_t**:

13081 CX {PTRDIFF_MIN} −65 536

13082 {PTRDIFF_MAX} +65 535

13083 • Limits of **sig_atomic_t**:

13084 {SIG_ATOMIC_MIN} See below.

13085 {SIG_ATOMIC_MAX} See below.

13086 • Limit of **size_t**:

13087 {SIZE_MAX} 65 535

13088 • Limits of **wchar_t**:

13089 {WCHAR_MIN} See below.

13090 {WCHAR_MAX} See below.

13091 • Limits of **wint_t**:

13092 {WINT_MIN} See below.

13093 {WINT_MAX} See below.

13094 If **sig_atomic_t** (see the **<signal.h>** header) is defined as a signed integer type, the value of
13095 {SIG_ATOMIC_MIN} shall be no greater than −127 and the value of {SIG_ATOMIC_MAX} shall
13096 be no less than 127; otherwise, **sig_atomic_t** shall be defined as an unsigned integer type, and
13097 the value of {SIG_ATOMIC_MIN} shall be 0 and the value of {SIG_ATOMIC_MAX} shall be no
13098 less than 255.

13099 If **wchar_t** (see the **<stddef.h>** header) is defined as a signed integer type, the value of
13100 {WCHAR_MIN} shall be no greater than −127 and the value of {WCHAR_MAX} shall be no less
13101 than 127; otherwise, **wchar_t** shall be defined as an unsigned integer type, and the value of
13102 {WCHAR_MIN} shall be 0 and the value of {WCHAR_MAX} shall be no less than 255.

13103 If **wint_t** (see the **<wchar.h>** header) is defined as a signed integer type, the value of
13104 {WINT_MIN} shall be no greater than −32 767 and the value of {WINT_MAX} shall be no less
13105 than 32 767; otherwise, **wint_t** shall be defined as an unsigned integer type, and the value of
13106 {WINT_MIN} shall be 0 and the value of {WINT_MAX} shall be no less than 65 535.

13107 **Macros for Integer Constant Expressions**

13108 The following macros expand to integer constant expressions suitable for initializing objects that
13109 have integer types corresponding to types defined in the **<stdint.h>** header. Each macro name
13110 corresponds to a similar type name listed under *Minimum-width integer types* and *Greatest-width*
13111 *integer types*.

13112 Each invocation of one of these macros shall expand to an integer constant expression suitable
13113 for use in **#if** preprocessing directives. The type of the expression shall have the same type as
13114 would an expression that is an object of the corresponding type converted according to the
13115 integer promotions. The value of the expression shall be that of the argument.

13116 The argument in any instance of these macros shall be an unsuffixed integer constant with a
13117 value that does not exceed the limits for the corresponding type.

13118 • Macros for minimum-width integer constant expressions

13119 The macro *INTN_C*(*value*) shall expand to an integer constant expression corresponding to
13120 the type **int_least***N***_t**. The macro *UINTN_C*(*value*) shall expand to an integer constant
13121 expression corresponding to the type **uint_least***N***_t**. For example, if **uint_least64_t** is a
13122 name for the type **unsigned long long**, then *UINT64_C*(0x123) might expand to the integer
13123 constant 0x123ULL.

13124 • Macros for greatest-width integer constant expressions

13125 The following macro expands to an integer constant expression having the value specified
13126 by its argument and the type **intmax_t**:

13127 INTMAX_C(*value*)

13128 The following macro expands to an integer constant expression having the value specified
13129 by its argument and the type **uintmax_t**:

13130 UINTMAX_C(*value*)

13131 **APPLICATION USAGE**

13132 None.

13133 **RATIONALE**

13134 The **<stdint.h>** header is a subset of the **<inttypes.h>** header more suitable for use in
13135 freestanding environments, which might not support the formatted I/O functions. In some
13136 environments, if the formatted conversion support is not wanted, using this header instead of
13137 the **<inttypes.h>** header avoids defining such a large number of macros.

13138 As a consequence of adding **int8_t**, the following are true:

13139 • A byte is exactly 8 bits.

13140 • {CHAR_BIT} has the value 8, {SCHAR_MAX} has the value 127, {SCHAR_MIN} has the
13141 value –128, and {UCHAR_MAX} has the value 255.

13142 Since the POSIX.1 standard explicitly requires 8-bit **char** with two's complement arithmetic, it is
13143 easier for application writers if the same two's complement guarantees are extended to all of the
13144 other standard integer types. Furthermore, in programming environments with a 32-bit **long**,
13145 some POSIX.1 interfaces, such as *mrand48*( ), cannot be implemented if **long** does not use a two's
13146 complement representation.

13147 **FUTURE DIRECTIONS**

13148 **typedef** names beginning with **int** or **uint** and ending with _t may be added to the types defined
13149 in the **<stdint.h>** header. Macro names beginning with INT or UINT and ending with _MAX,
13150 _MIN, or _C may be added to the macros defined in the **<stdint.h>** header.

13151 **SEE ALSO**

13152 **<inttypes.h>**, **<signal.h>**, **<stddef.h>**, **<wchar.h>**

13153 XSH Section 2.2 (on page 496)

13154 **CHANGE HISTORY**

13155 First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

13156 ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is applied.

13157 **Issue 7**

13158 ISO/IEC 9899: 1999 standard, Technical Corrigendum 3 #40 is applied.

13159 SD5-XBD-ERN-67 is applied.

13160 **Issue 8**

13161 Austin Group Defect 1108 is applied, changing the maximum allowed value for all signed
13162 integer minimum limits.

13163 Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899: 2018 standard.

13164 Austin Group Defect 1330 is applied, changing ``_V7_'' to ``_V8_''.

NAME

stdio.h — standard buffered input/output

SYNOPSIS

```
#include <stdio.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to enable the visibility of these symbols in this header.

The **<stdio.h>** header shall define the following data types through **typedef**:

FILE A type containing information about a file. The **FILE** type may be an incomplete type.

fpos_t A complete object type, other than an array type, capable of recording all the information needed to specify uniquely every position within a file.

off_t As described in **<sys/types.h>**.

size_t As described in **<stddef.h>**.

CX **ssize_t** As described in **<sys/types.h>**.

CX **va_list** As described in **<stdarg.h>**.

The **<stdio.h>** header shall define the following macros which shall expand to integer constant expressions:

CX BUFSIZ Size of **<stdio.h>** buffers. This shall expand to a positive value.

CX L_ctermid Maximum size of character array to hold *ctermid*() output.

OB L_tmpnam Maximum size of character array to hold *tmpnam*() output.

The **<stdio.h>** header shall define the following macros which shall expand to integer constant expressions with distinct values:

_IOFBF Input/output fully buffered.

_IOLBF Input/output line buffered.

_IONBF Input/output unbuffered.

The **<stdio.h>** header shall define the following macros which shall expand to integer constant expressions with distinct values:

SEEK_CUR Seek relative to current position.

SEEK_END Seek relative to end-of-file.

SEEK_SET Seek relative to start-of-file.

The **<stdio.h>** header shall define the following macros which shall expand to integer constant expressions denoting implementation limits:

{FILENAME_MAX} Maximum size in bytes of the longest pathname that the implementation guarantees can be opened.

{FOPEN_MAX} Number of streams which the implementation guarantees can be open simultaneously. The value is at least eight.

13203 OB {TMP_MAX} Minimum number of unique filenames generated by *tmpnam*(). Maximum number of times an application can call *tmpnam*() reliably. The value of {TMP_MAX} is at least 25.

13206 OB XSI On XSI-conformant systems, the value of {TMP_MAX} is at least 10 000.

13207 The **<stdio.h>** header shall define the following macro which shall expand to an integer constant
13208 expression with type **int** and a negative value:

13209 EOF End-of-file return value.

13210 The **<stdio.h>** header shall define NULL as described in **<stddef.h>**.

13211 The **<stdio.h>** header shall define the following macros which shall expand to expressions of
13212 type ``pointer to **FILE**'' that point to the **FILE** objects associated, respectively, with the standard
13213 error, input, and output streams:

13214 *stderr* Standard error output stream.

13215 *stdin* Standard input stream.

13216 *stdout* Standard output stream.

13217 The following shall be declared as functions and may also be defined as macros. Function
13218 prototypes shall be provided.

```
13219        void     clearerr(FILE *);
13220 CX     char    *ctermid(char *);
13221        int      dprintf(int, const char *restrict, ...)
13222        int      fclose(FILE *);
13223 CX     FILE    *fdopen(int, const char *);
13224        int      feof(FILE *);
13225        int      ferror(FILE *);
13226        int      fflush(FILE *);
13227        int      fgetc(FILE *);
13228        int      fgetpos(FILE *restrict, fpos_t *restrict);
13229        char    *fgets(char *restrict, int, FILE *restrict);
13230 CX     int      fileno(FILE *);
13231        void     flockfile(FILE *);
13232        FILE    *fmemopen(void *restrict, size_t, const char *restrict);
13233        FILE    *fopen(const char *restrict, const char *restrict);
13234        int      fprintf(FILE *restrict, const char *restrict, ...);
13235        int      fputc(int, FILE *);
13236        int      fputs(const char *restrict, FILE *restrict);
13237        size_t   fread(void *restrict, size_t, size_t, FILE *restrict);
13238        FILE    *freopen(const char *restrict, const char *restrict,
13239                     FILE *restrict);
13240        int      fscanf(FILE *restrict, const char *restrict, ...);
13241        int      fseek(FILE *, long, int);
13242 CX     int      fseeko(FILE *, off_t, int);
13243        int      fsetpos(FILE *, const fpos_t *);
13244        long     ftell(FILE *);
13245 CX     off_t    ftello(FILE *);
13246        int      ftrylockfile(FILE *);
13247        void     funlockfile(FILE *);
13248        size_t   fwrite(const void *restrict, size_t, size_t, FILE *restrict);
13249        int      getc(FILE *);
```



```
13250          int       getchar(void);
13251   CX     int       getc_unlocked(FILE *);
13252          int       getchar_unlocked(void);
13253          ssize_t   getdelim(char **restrict, size_t *restrict, int,
13254                        FILE *restrict);
13255          ssize_t   getline(char **restrict, size_t *restrict, FILE *restrict);
13256          FILE     *open_memstream(char **, size_t *);
13257          int       pclose(FILE *);
13258          void      perror(const char *);
13259   CX     FILE     *popen(const char *, const char *);
13260          int       printf(const char *restrict, ...);
13261          int       putc(int, FILE *);
13262          int       putchar(int);
13263   CX     int       putc_unlocked(int, FILE *);
13264          int       putchar_unlocked(int);
13265          int       puts(const char *);
13266          int       remove(const char *);
13267          int       rename(const char *, const char *);
13268   CX     int       renameat(int, const char *, int, const char *);
13269          void      rewind(FILE *);
13270          int       scanf(const char *restrict, ...);
13271          void      setbuf(FILE *restrict, char *restrict);
13272          int       setvbuf(FILE *restrict, char *restrict, int, size_t);
13273          int       snprintf(char *restrict, size_t, const char *restrict, ...);
13274          int       sprintf(char *restrict, const char *restrict, ...);
13275          int       sscanf(const char *restrict, const char *restrict, ...);
13276          FILE     *tmpfile(void);
13277   OB     char     *tmpnam(char *);
13278          int       ungetc(int, FILE *);
13279   CX     int       vdprintf(int, const char *restrict, va_list);
13280          int       vfprintf(FILE *restrict, const char *restrict, va_list);
13281          int       vfscanf(FILE *restrict, const char *restrict, va_list);
13282          int       vprintf(const char *restrict, va_list);
13283          int       vscanf(const char *restrict, va_list);
13284          int       vsnprintf(char *restrict, size_t, const char *restrict,
13285                        va_list);
13286          int       vsprintf(char *restrict, const char *restrict, va_list);
13287          int       vsscanf(const char *restrict, const char *restrict, va_list);
```

13288 CX Inclusion of the **<stdio.h>** header may also make visible all symbols from **<stddef.h>**.

13289 **APPLICATION USAGE**

13290 Since standard I/O streams may use an underlying file descriptor to access the file associated
13291 with a stream, application developers need to be aware that {FOPEN_MAX} streams may not be
13292 available if file descriptors are being used to access files that are not associated with streams.

13293 Since the latest revision of the ISO C standard allows **FILE** to be an incomplete type (and POSIX
13294 also allows it), portable applications can no longer allocate or copy an object of type **FILE**; only
13295 pointers to objects of type **FILE** can be allocated.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<stdarg.h>, **<stddef.h>**, **<sys/types.h>**

XSH Section 2.2 (on page 496), *clearerr*( ), *ctermid*( ), *fclose*( ), *fdopen*( ), *feof*( ), *ferror*( ), *fflush*( ), *fgetc*( ), *fgetpos*( ), *fgets*( ), *fileno*( ), *flockfile*( ), *fmemopen*( ), *fopen*( ), *fprintf*( ), *fputc*( ), *fputs*( ), *fread*( ), *freopen*( ), *fscanf*( ), *fseek*( ), *fsetpos*( ), *ftell*( ), *fwrite*( ), *getc*( ), *getchar*( ), *getc_unlocked*( ), *getdelim*( ), *getopt*( ), *open_memstream*( ), *pclose*( ), *perror*( ), *popen*( ), *putc*( ), *putchar*( ), *puts*( ), *remove*( ), *rename*( ), *rewind*( ), *setbuf*( ), *setvbuf*( ), *stdin*, *system*( ), *tmpfile*( ), *tmpnam*( ), *ungetc*( ), *vfprintf*( ), *vfscanf*( )

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Large File System extensions are added.

The constant L_cuserid and the external variables *optarg*, *opterr*, *optind*, and *optopt* are marked as extensions and LEGACY.

The *cuserid*( ) and *getopt*( ) functions are marked LEGACY.

Issue 6

The constant L_cuserid and the external variables *optarg*, *opterr*, *optind*, and *optopt* are removed as they were previously marked LEGACY.

The *cuserid*( ), *getopt*( ), and *getw*( ) functions are removed as they were previously marked LEGACY.

Several functions are marked as part of the Thread-Safe Functions option.

This reference page is updated to align with the ISO/IEC 9899:1999 standard. Note that the description of the **fpos_t** type is now explicitly updated to exclude array types.

Extensions beyond the ISO C standard are marked.

Issue 7

Austin Group Interpretation 1003.1-2001 #172 is applied, adding rationale about a conflict for the definition of {TMP_MAX} with the ISO C standard.

SD5-XBD-ERN-99 is applied, adding APPLICATION USAGE.

The *dprintf*( ), *fmemopen*( ), *getdelim*( ), *getline*( ), *open_memstream*( ), and *vdprintf*( ) functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The *renameat*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The *gets*( ), *tmpnam*( ), and *tempnam*( ) functions and the L_tmpnam macro are marked obsolescent.

This reference page is clarified with respect to macros and symbolic constants, and a declaration for the **off_t** type is added.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0065 [291,427] is applied.

Issue 8

Austin Group Defect 1054 is applied, allowing **FILE** to be an incomplete type.

Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899: 2018 standard.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

NAME

stdlib.h — standard library definitions

SYNOPSIS

```
#include <stdlib.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to enable the visibility of these symbols in this header.

The **<stdlib.h>** header shall define the following macros which shall expand to integer constant expressions:

CX EXIT_FAILURE Unsuccessful termination for *exit*( ); the value shall be between 1 and 125 inclusive.

EXIT_SUCCESS Successful termination for *exit*( ); the value shall be 0.

{RAND_MAX} Maximum value returned by *rand*( ); at least 32 767.

The **<stdlib.h>** header shall define the following macro which shall expand to a positive integer expression with type **size_t**:

{MB_CUR_MAX} Maximum number of bytes in a character specified by the current locale (category *LC_CTYPE*).

CX In the POSIX locale the value of {MB_CUR_MAX} shall be 1.

The **<stdlib.h>** header shall define NULL as described in **<stddef.h>**.

The **<stdlib.h>** header shall define the following data types through **typedef**:

div_t Structure type returned by the *div*( ) function.

ldiv_t Structure type returned by the *ldiv*( ) function.

lldiv_t Structure type returned by the *lldiv*( ) function.

size_t As described in **<stddef.h>**.

wchar_t As described in **<stddef.h>**.

CX In addition, the **<stdlib.h>** header shall define the following symbolic constants and macros as described in **<sys/wait.h>**:

WCOREDUMP
WEXITSTATUS
WIFEXITED
WIFSIGNALED
WIFSTOPPED
WNOHANG
WSTOPSIG
WTERMSIG
WUNTRACED

CX The **<stdlib.h>** header shall define the following symbolic constants as described in **<fcntl.h>**:

O_APPEND
O_CLOEXEC
O_CLOFORK

13383 SIO O_DSYNC
13384 XSI O_NOCTTY
13385 O_RDWR
13386 SIO O_RSYNC
13387 CX O_SYNC

13388 The following shall be declared as functions and may also be defined as macros. Function
13389 prototypes shall be provided.

```
13390       _Noreturn void  _Exit(int);
13391 XSI   long            a64l(const char *);
13392       _Noreturn void  abort(void);
13393       int             abs(int);
13394       void           *aligned_alloc(size_t, size_t);
13395       int             at_quick_exit(void (*)(void));
13396       int             atexit(void (*)(void));
13397       double          atof(const char *);
13398       int             atoi(const char *);
13399       long            atol(const char *);
13400       long long       atoll(const char *);
13401       void           *bsearch(const void *, const void *, size_t, size_t,
13402                           int (*)(const void *, const void *));
13403       void           *calloc(size_t, size_t);
13404       div_t           div(int, int);
13405 XSI   double          drand48(void);
13406       double          erand48(unsigned short [3]);
13407       _Noreturn void  exit(int);
13408       void            free(void *);
13409       char           *getenv(const char *);
13410 CX    int             getsubopt(char **restrict, char *const *restrict,
13411                           char **restrict);
13412 XSI   int             grantpt(int);
13413       char           *initstate(unsigned, char *, size_t);
13414       long            jrand48(unsigned short [3]);
13415       char           *l64a(long);
13416       long            labs(long);
13417 XSI   void            lcong48(unsigned short [7]);
13418       ldiv_t          ldiv(long, long);
13419       long long       llabs(long long);
13420       lldiv_t         lldiv(long long, long long);
13421 XSI   long            lrand48(void);
13422       void           *malloc(size_t);
13423       int             mblen(const char *, size_t);
13424       size_t          mbstowcs(wchar_t *restrict, const char *restrict,
13425                           size_t);
13426       int             mbtowc(wchar_t *restrict, const char *restrict, size_t);
13427 CX    char           *mkdtemp(char *);
13428       int             mkostemp(char *, int);
13429       int             mkstemp(char *);
13430 XSI   long            mrand48(void);
13431       long            nrand48(unsigned short [3]);
```



```
ADV     int              posix_memalign(void **, size_t, size_t);
XSI     int              posix_openpt(int);
        char            *ptsname(int);
        int              ptsname_r(int, char *, size_t);
        int              putenv(char *);
        void             qsort(void *, size_t, size_t, int (*)(const void *,
                             const void *));
        _Noreturn void   quick_exit(int);
CX      void             qsort_r(void *, size_t, size_t, int (*)(const void *,
                             const void *, void *), void *);
        int              rand(void);
XSI     long             random(void);
        void            *realloc(void *, size_t);
CX      void            *reallocarray(void *, size_t, size_t);
        char            *realpath(const char *restrict, char *restrict);
        char            *secure_getenv(const char *);
XSI     unsigned short  *seed48(unsigned short [3]);
CX      int              setenv(const char *, const char *, int);
OB XSI  void             setkey(const char *);
XSI     char             *setstate(char *);
        void             srand(unsigned);
XSI     void             srand48(long);
        void             srandom(unsigned);
        double           strtod(const char *restrict, char **restrict);
        float            strtof(const char *restrict, char **restrict);
        long             strtol(const char *restrict, char **restrict, int);
        long double      strtold(const char *restrict, char **restrict);
        long long        strtoll(const char *restrict, char **restrict, int);
        unsigned long    strtoul(const char *restrict, char **restrict, int);
        unsigned long long
                         strtoull(const char *restrict, char **restrict, int);
        int              system(const char *);
XSI     int              unlockpt(int);
CX      int              unsetenv(const char *);
        size_t           wcstombs(char *restrict, const wchar_t *restrict,
                             size_t);
        int              wctomb(char *, wchar_t);
```

CX Inclusion of the **<stdlib.h>** header may also make visible all symbols from **<fcntl.h>**, **<limits.h>**, **<math.h>**, **<stddef.h>**, and **<sys/wait.h>**.

APPLICATION USAGE

None.

RATIONALE

The ISO C standard requires that `exit(EXIT_FAILURE)` returns ``unsuccessful termination status'' to the host environment. In a POSIX host environment this means that the lower 8 bits of EXIT_FAILURE must have at least one bit set. The standard developers decided to further restrict the allowed values for the following reasons:

- Exit statuses of 126 and greater are ambiguous in certain circumstances because they have special meanings in the shell (see XCU Section 2.8.2 (on page 2499) and the EXIT STATUS section of *sh*).

- The *xargs* utility quits when a command execution exits with status 255 (see XCU *xargs*).
- Calling *exit*( ) with a value greater than 255 or less than 0 is something that only programs which are specifically designed to have their exit status obtained by *waitid*( ) should do (since it does not truncate the exit status to 8 bits). ``Pure ISO C'' programs that call `exit(EXIT_FAILURE)` do not meet this design criterion.

The requirement that the value of EXIT_SUCCESS is 0 is not shaded CX because this matches the requirement in the ISO C standard that `exit(EXIT_SUCCESS)` returns ``successful termination status'' to the host environment (when the host environment is a POSIX implementation).

FUTURE DIRECTIONS

None.

SEE ALSO

<limits.h>, **<math.h>**, **<stddef.h>**, **<sys/types.h>**, **<sys/wait.h>**

XSH Section 2.2 (on page 496), *_Exit*( ), *a64l*( ), *abort*( ), *abs*( ), *atexit*( ), *atof*( ), *atoi*( ), *atol*( ), *bsearch*( ), *calloc*( ), *div*( ), *drand48*( ), *exit*( ), *free*( ), *getenv*( ), *getsubopt*( ), *grantpt*( ), *initstate*( ), *labs*( ), *ldiv*( ), *malloc*( ), *mblen*( ), *mbstowcs*( ), *mbtowc*( ), *mkdtemp*( ), *posix_memalign*( ), *posix_openpt*( ), *ptsname*( ), *putenv*( ), *qsort*( ), *rand*( ), *realloc*( ), *realpath*( ), *setenv*( ), *setkey*( ), *strtod*( ), *strtol*( ), *strtoul*( ), *system*( ), *unlockpt*( ), *unsetenv*( ), *waitid*( ), *wcstombs*( ), *wctomb*( )

CHANGE HISTORY

First released in Issue 3.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

The *ttyslot*( ) and *valloc*( ) functions are marked LEGACY.

The type of the third argument to *initstate*( ) is changed from **int** to **size_t**. The type of the return value from *setstate*( ) is changed from **char** to **char** *, and the type of the first argument is changed from **char** * to **const char** *.

Issue 6

The Open Group Corrigendum U021/1 is applied, correcting the prototype for *realpath*( ) to be consistent with the reference page.

The Open Group Corrigendum U028/13 is applied, correcting the prototype for *putenv*( ) to be consistent with the reference page.

The *rand_r*( ) function is marked as part of the Thread-Safe Functions option.

Function prototypes for *setenv*( ) and *unsetenv*( ) are added.

The *posix_memalign*( ) function is added for alignment with IEEE Std 1003.1d-1999.

This reference page is updated to align with the ISO/IEC 9899: 1999 standard.

The *ecvt*( ), *fcvt*( ), *gcvt*( ), and *mktemp*( ) functions are marked LEGACY.

The *ttyslot*( ) and *valloc*( ) functions are removed as they were previously marked LEGACY.

Extensions beyond the ISO C standard are marked.

Issue 7

SD5-XBD-ERN-79 and SD5-XBD-ERN-105 are applied.

The LEGACY functions are removed.

The *mkdtemp*( ) function is added from The Open Group Technical Standard, 2006, Extended API

Set Part 1.

The *rand_r*( ) function is marked obsolescent.

This reference page is clarified with respect to macros and symbolic constants.

The type of the first argument to *setstate*( ) is changed from **const char** * to **char** *.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0066 [197] is applied.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0075 [663] is applied.

Issue 8

Austin Group Defect 411 is applied, adding *mkostemp*( ).

Austin Group Defect 444 is applied, adding the **restrict** keyword to the *getsubopt*( ) prototype.

Austin Group Defect 508 is applied, adding the *ptsname_r*( ) function.

Austin Group Defects 593 and 1350 are applied, adding some O_* symbolic constants and allowing **<stdlib.h>** to make visible all symbols from **<fcntl.h>**.

Austin Group Defect 900 is applied, adding the *qsort_r*( ) function.

Austin Group Defect 922 is applied, adding the *secure_getenv*( ) function.

Austin Group Defect 1141 is applied, adding WCOREDUMP.

Austin Group Defect 1192 is applied, marking the *setkey*( ) function as obsolescent.

Austin Group Defect 1218 is applied, adding *reallocarray*( ).

Austin Group Defect 1229 is applied, changing the descriptions of EXIT_FAILURE and EXIT_SUCCESS, and the RATIONALE section.

Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899: 2018 standard.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1629 is applied, changing the RATIONALE section.

Austin Group Defect 1663 is applied, removing XSI shading from *realpath*( ).

NAME

stdnoreturn.h — noreturn macro

SYNOPSIS

```
#include <stdnoreturn.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The **<stdnoreturn.h>** header shall define the macro noreturn which shall expand to **_Noreturn**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

None.

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

string.h — string operations

SYNOPSIS

```
#include <string.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to enable the visibility of these symbols in this header.

The **<string.h>** header shall define NULL and **size_t** as described in **<stddef.h>**.

CX The **<string.h>** header shall define the **locale_t** type as described in **<locale.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

```
XSI void    *memccpy(void *restrict, const void *restrict, int, size_t);
    void    *memchr(const void *, int, size_t);
    int      memcmp(const void *, const void *, size_t);
    void    *memcpy(void *restrict, const void *restrict, size_t);
CX  void    *memmem(const void *, size_t, const void *, size_t);
    void    *memmove(void *, const void *, size_t);
    void    *memset(void *, int, size_t);
CX  char    *stpcpy(char *restrict, const char *restrict);
    char    *stpncpy(char *restrict, const char *restrict, size_t);
    char    *strcat(char *restrict, const char *restrict);
    char    *strchr(const char *, int);
    int      strcmp(const char *, const char *);
    int      strcoll(const char *, const char *);
CX  int      strcoll_l(const char *, const char *, locale_t);
    char    *strcpy(char *restrict, const char *restrict);
    size_t   strcspn(const char *, const char *);
CX  char    *strdup(const char *);
    char    *strerror(int);
CX  char    *strerror_l(int, locale_t);
    int      strerror_r(int, char *, size_t);
    size_t   strlcat(char *restrict, const char *restrict, size_t);
    size_t   strlcpy(char *restrict, const char *restrict, size_t);
    size_t   strlen(const char *);
    char    *strncat(char *restrict, const char *restrict, size_t);
    int      strncmp(const char *, const char *, size_t);
    char    *strncpy(char *restrict, const char *restrict, size_t);
CX  char    *strndup(const char *, size_t);
    size_t   strnlen(const char *, size_t);
    char    *strpbrk(const char *, const char *);
    char    *strrchr(const char *, int);
CX  char    *strsignal(int);
    size_t   strspn(const char *, const char *);
    char    *strstr(const char *, const char *);
    char    *strtok(char *restrict, const char *restrict);
CX  char    *strtok_r(char *restrict, const char *restrict, char **restrict);
    size_t   strxfrm(char *restrict, const char *restrict, size_t);
```



```
CX  size_t   strxfrm_l(char *restrict, const char *restrict,
                size_t, locale_t);
```

CX Inclusion of the **<string.h>** header may also make visible all symbols from **<stddef.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<locale.h>, **<stddef.h>**, **<sys/types.h>**

XSH Section 2.2 (on page 496), *memccpy*( ), *memchr*( ), *memcmp*( ), *memcpy*( ), *memmem*( ), *memmove*( ), *memset*( ), *strcat*( ), *strchr*( ), *strcmp*( ), *strcoll*( ), *strcpy*( ), *strcspn*( ), *strdup*( ), *strerror*( ), *strlcat*( ), *strlen*( ), *strncat*( ), *strncmp*( ), *strncpy*( ), *strpbrk*( ), *strrchr*( ), *strsignal*( ), *strspn*( ), *strstr*( ), *strtok*( ), *strxfrm*( )

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The *strtok_r*( ) function is marked as part of the Thread-Safe Functions option.

This reference page is updated to align with the ISO/IEC 9899: 1999 standard.

The *strerror_r*( ) function is added in response to IEEE PASC Interpretation 1003.1c #39.

Issue 7

SD5-XBD-ERN-15 is applied, correcting the prototype for the *strerror_r*( ) function.

The *stpcpy*( ), *stpncpy*( ), *strndup*( ), *strnlen*( ), and *strsignal*( ) functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The *strcoll_l*( ), *strerror_l*( ), and *strxfrm_l*( ) functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

This reference page is clarified with respect to macros and symbolic constants, and a declaration for the **locale_t** type is added.

Issue 8

Austin Group Defect 986 is applied, adding *strlcat*( ) and *strlcpy*( ).

Austin Group Defect 1061 is applied, adding *memmem*( ).

NAME

strings.h — string operations

SYNOPSIS

```
#include <strings.h>
```

DESCRIPTION

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

XSI
```
int    ffs(int);
int    ffsl(long);
int    ffsll(long long);
int    strcasecmp(const char *, const char *);
int    strcasecmp_l(const char *, const char *, locale_t);
int    strncasecmp(const char *, const char *, size_t);
int    strncasecmp_l(const char *, const char *, size_t, locale_t);
```

The **<strings.h>** header shall define the **locale_t** type as described in **<locale.h>**.

The **<strings.h>** header shall define the **size_t** type as described in **<sys/types.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<locale.h>, **<sys/types.h>**

XSH *ffs*( ), *strcasecmp*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 6

The Open Group Corrigendum U021/2 is applied, correcting the prototype for *index*( ) to be consistent with the reference page.

The *bcmp*( ), *bcopy*( ), *bzero*( ), *index*( ), and *rindex*( ) functions are marked LEGACY.

Issue 7

SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

The LEGACY functions are removed.

The **<strings.h>** header is moved from the XSI option to the Base.

The *strcasecmp_l*( ) and *strncasecmp_l*( ) functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

A declaration for the **locale_t** type is added.

Issue 8

Austin Group Defect 617 is applied, adding *ffsl*( ) and *ffsll*( ).

13686 **NAME**

13687 sys/ipc.h — XSI interprocess communication access structure

13688 **SYNOPSIS**

13689 XSI
```
#include <sys/ipc.h>
```

13690 **DESCRIPTION**

13691 The **<sys/ipc.h>** header is used by three mechanisms for XSI interprocess communication (IPC):
13692 messages, semaphores, and shared memory. All use a common structure type, **ipc_perm**, to pass
13693 information used in determining permission to perform an IPC operation.

13694 The **<sys/ipc.h>** header shall define the **ipc_perm** structure, which shall include the following
13695 members:

13696 `uid_t` `uid` Owner's user ID.
13697 `gid_t` `gid` Owner's group ID.
13698 `uid_t` `cuid` Creator's user ID.
13699 `gid_t` `cgid` Creator's group ID.
13700 `mode_t` `mode` Read/write permission.

13701 The **<sys/ipc.h>** header shall define the **uid_t**, **gid_t**, **mode_t**, and **key_t** types as described in
13702 **<sys/types.h>**.

13703 The **<sys/ipc.h>** header shall define the following symbolic constants.

13704 Mode bits:

13705 IPC_CREAT Create entry if key does not exist.

13706 IPC_EXCL Fail if key exists.

13707 IPC_NOWAIT Error if request would need to wait.

13708 Keys:

13709 IPC_PRIVATE Private key.

13710 Control commands:

13711 IPC_RMID Remove identifier.

13712 IPC_SET Set options.

13713 IPC_STAT Get options.

13714 The following shall be declared as a function and may also be defined as a macro. A function
13715 prototype shall be provided.

13716
```
key_t  ftok(const char *, int);
```

13717 **APPLICATION USAGE**

13718 None.

13719 **RATIONALE**

13720 None.

13721 **FUTURE DIRECTIONS**

13722 None.

13723 **SEE ALSO**

13724 **<sys/types.h>**

13725 XSH *ftok*( )

13726 **CHANGE HISTORY**

13727 First released in Issue 2. Derived from System V Release 2.0.

13728 **Issue 7**

13729 This reference page is clarified with respect to macros and symbolic constants.

13730 **Issue 8**

13731 The description of IPC_NOWAIT is updated to eliminate the use of ``must''.

13732 **NAME**

13733 sys/mman.h — memory management declarations

13734 **SYNOPSIS**

13735 `#include <sys/mman.h>`

13736 **DESCRIPTION**

13737 The **<sys/mman.h>** header shall define the following symbolic constants for use as protection
13738 options:

13739 PROT_EXEC Page can be executed.

13740 PROT_NONE Page cannot be accessed.

13741 PROT_READ Page can be read.

13742 PROT_WRITE Page can be written.

13743 The **<sys/mman.h>** header shall define the following symbolic constants for use as flag options:

13744 MAP_ANON Synonym for MAP_ANONYMOUS. MAP_ANON shall have the same
13745 value as MAP_ANONYMOUS.

13746 MAP_ANONYMOUS Map anonymous memory.

13747 MAP_FIXED Interpret *addr* exactly.

13748 MAP_PRIVATE Changes are private.

13749 MAP_SHARED Share changes.

13750 XSI|SIO The **<sys/mman.h>** header shall define the following symbolic constants for the *msync*( )
13751 function:

13752 MS_ASYNC Perform asynchronous writes.

13753 MS_INVALIDATE Invalidate mappings.

13754 MS_SYNC Perform synchronous writes.

13755 ML The **<sys/mman.h>** header shall define the following symbolic constants for the *mlockall*( )
13756 function:

13757 MCL_CURRENT Lock currently mapped pages.

13758 MCL_FUTURE Lock pages that become mapped.

13759 The **<sys/mman.h>** header shall define the symbolic constant MAP_FAILED which shall have
13760 type **void** * and shall be used to indicate a failure from the *mmap*( ) function .

13761 ADV If the Advisory Information option is supported, the **<sys/mman.h>** header shall define
13762 symbolic constants for the *advice* argument to the *posix_madvise*( ) function as follows:

13763 POSIX_MADV_DONTNEED
13764 The application expects that it will not access the specified range in the near future.

13765 POSIX_MADV_NORMAL
13766 The application has no advice to give on its behavior with respect to the specified range. It
13767 is the default characteristic if no advice is given for a range of memory.

POSIX_MADV_RANDOM
The application expects to access the specified range in a random order.

POSIX_MADV_SEQUENTIAL
The application expects to access the specified range sequentially from lower addresses to higher addresses.

POSIX_MADV_WILLNEED
The application expects to access the specified range in the near future.

TYM The **<sys/mman.h>** header shall define the following symbolic constants for use as flags for the *posix_typed_mem_open*( ) function:

POSIX_TYPED_MEM_ALLOCATE
Allocate on *mmap*( ).

POSIX_TYPED_MEM_ALLOCATE_CONTIG
Allocate contiguously on *mmap*( ).

POSIX_TYPED_MEM_MAP_ALLOCATABLE
Map on *mmap*( ), without affecting allocatability.

The **<sys/mman.h>** header shall define the **mode_t**, **off_t**, and **size_t** types as described in **<sys/types.h>**.

TYM The **<sys/mman.h>** header shall define the **posix_typed_mem_info** structure, which shall include at least the following member:

`size_t`	`posix_tmi_length`	Maximum length which may be allocated from a typed memory object.

The **<sys/mman.h>** header shall define the following symbolic constants as described in **<fcntl.h>**:

SHM|TYM O_RDONLY
O_RDWR
TYM O_WRONLY
O_CLOEXEC
O_CLOFORK
SHM O_CREAT
O_EXCL
O_TRUNC

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
MLR  int    mlock(const void *, size_t);
ML   int    mlockall(int);
     void  *mmap(void *, size_t, int, int, int, off_t);
     int    mprotect(void *, size_t, int);
XSI|SIO int msync(void *, size_t, int);
MLR  int    munlock(const void *, size_t);
ML   int    munlockall(void);
     int    munmap(void *, size_t);
```



```
ADV     int     posix_madvise(void *, size_t, int);
TYM     int     posix_mem_offset(const void *restrict, size_t, off_t *restrict,
                    size_t *restrict, int *restrict);
        int     posix_typed_mem_get_info(int, struct posix_typed_mem_info *);
        int     posix_typed_mem_open(const char *, int, int);
SHM     int     shm_open(const char *, int, mode_t);
        int     shm_unlink(const char *);
```

Inclusion of the **<sys/mman.h>** header may make visible all symbols from the **<fcntl.h>** header.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>

XSH *mlock*( ), *mlockall*( ), *mmap*( ), *mprotect*( ), *msync*( ), *munmap*( ), *posix_madvise*( ), *posix_mem_offset*( ), *posix_typed_mem_get_info*( ), *posix_typed_mem_open*( ), *shm_open*( ), *shm_unlink*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Updated for alignment with the POSIX Realtime Extension.

Issue 6

The **<sys/mman.h>** header is marked as dependent on support for either the Memory Mapped Files, Process Memory Locking, or Shared Memory Objects options.

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The TYM margin code is added to the list of margin codes for the **<sys/mman.h>** header line, as well as for other lines.
- The POSIX_TYPED_MEM_ALLOCATE, POSIX_TYPED_MEM_ALLOCATE_CONTIG, and POSIX_TYPED_MEM_MAP_ALLOCATABLE flags are added.
- The **posix_tmi_length** structure is added.
- The *posix_mem_offset*( ), *posix_typed_mem_get_info*( ), and *posix_typed_mem_open*( ) functions are added.

The **restrict** keyword is added to the prototype for *posix_mem_offset*( ).

IEEE PASC Interpretation 1003.1 #102 is applied, adding the prototype for *posix_madvise*( ).

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/16 is applied, correcting margin code and shading errors for the *mlock*( ) and *munlock*( ) functions.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/34 is applied, changing the margin code for the *mmap*( ) function from MF|SHM to MC3 (notation for MF|SHM|TYM).

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/36 is applied, changing the margin code

for the *munmap*( ) function from MF|SHM to MC3 (notation for MF|SHM|TYM).

Issue 7

SD5-XBD-ERN-5 is applied, rewriting the DESCRIPTION.

Functionality relating to the Memory Protection and Memory Mapped Files options is moved to the Base.

This reference page is clarified with respect to macros and symbolic constants.

Issue 8

Austin Group Defect 593 is applied, adding O_RDONLY, O_RDWR, O_WRONLY, O_CLOEXEC, O_CLOFORK, O_CREAT, O_EXCL, and O_TRUNC, and allowing **<sys/mman.h>** to make visible all symbols from **<fcntl.h>**.

Austin Group Defect 850 is applied, adding MAP_ANON and MAP_ANONYMOUS.

NAME

sys/msg.h — XSI message queue structures

SYNOPSIS

XSI
```
#include <sys/msg.h>
```

DESCRIPTION

The **<sys/msg.h>** header shall define the following data types through **typedef**:

msgqnum_t Used for the number of messages in the message queue.

msglen_t Used for the number of bytes allowed in a message queue.

These types shall be unsigned integer types that are able to store values at least as large as a type **unsigned short**.

The **<sys/msg.h>** header shall define the following symbolic constant as a message operation flag:

MSG_NOERROR No error if big message.

The **<sys/msg.h>** header shall define the **msqid_ds** structure, which shall include the following members:

`struct ipc_perm`	`msg_perm`	Operation permission structure.
`msgqnum_t`	`msg_qnum`	Number of messages currently on queue.
`msglen_t`	`msg_qbytes`	Maximum number of bytes allowed on queue.
`pid_t`	`msg_lspid`	Process ID of last *msgsnd*( ).
`pid_t`	`msg_lrpid`	Process ID of last *msgrcv*( ).
`time_t`	`msg_stime`	Time of last *msgsnd*( ).
`time_t`	`msg_rtime`	Time of last *msgrcv*( ).
`time_t`	`msg_ctime`	Time of last change.

The **<sys/msg.h>** header shall define the **pid_t**, **size_t**, **ssize_t**, and **time_t** types as described in **<sys/types.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int     msgctl(int, int, struct msqid_ds *);
int     msgget(key_t, int);
ssize_t msgrcv(int, void *, size_t, long, int);
int     msgsnd(int, const void *, size_t, int);
```

In addition, the **<sys/msg.h>** header shall include the **<sys/ipc.h>** header.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/ipc.h>, **<sys/types.h>**

XSH *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( )

13902 **CHANGE HISTORY**

13903 First released in Issue 2. Derived from System V Release 2.0.

13904 **Issue 7**

13905 Austin Group Interpretation 1003.1-2001 #179 is applied.

13906 This reference page is clarified with respect to macros and symbolic constants.

13907 **NAME**

13908 sys/resource.h — definitions for resource operations

13909 **SYNOPSIS**

13910 `#include <sys/resource.h>`

13911 **DESCRIPTION**

13912 XSI The **<sys/resource.h>** header shall define the following symbolic constants as possible values of
13913 the *which* argument of *getpriority*( ) and *setpriority*( ):

13914 PRIO_PROCESS	Identifies the *who* argument as a process ID.
13915 PRIO_PGRP	Identifies the *who* argument as a process group ID.
13916 PRIO_USER	Identifies the *who* argument as a user ID.

13917 The **<sys/resource.h>** header shall define the following type through **typedef**:

13918 **rlim_t**	Unsigned integer type used for limit values.

13919 The **<sys/resource.h>** header shall define the following symbolic constants, which shall have
13920 values suitable for use in **#if** preprocessing directives:

13921 RLIM_INFINITY	A value of **rlim_t** indicating no limit.
13922 RLIM_SAVED_MAX	A value of type **rlim_t** indicating an unrepresentable saved hard limit.
13924 RLIM_SAVED_CUR	A value of type **rlim_t** indicating an unrepresentable saved soft limit.

13925 On implementations where all resource limits are representable in an object of type **rlim_t**,
13926 RLIM_SAVED_MAX and RLIM_SAVED_CUR need not be distinct from RLIM_INFINITY.

13927 XSI The **<sys/resource.h>** header shall define the following symbolic constants as possible values of
13928 the *who* parameter of *getrusage*( ):

13929 RUSAGE_SELF	Returns information about the current process.
13930 RUSAGE_CHILDREN	Returns information about children of the current process.

13931 The **<sys/resource.h>** header shall define the **rlimit** structure, which shall include at least the
13932 following members:

13933 `rlim_t rlim_cur`	The current (soft) limit.
13934 `rlim_t rlim_max`	The hard limit.

13935 XSI The **<sys/resource.h>** header shall define the **rusage** structure, which shall include at least the
13936 following members:

13937 `struct timeval ru_utime`	User time used.
13938 `struct timeval ru_stime`	System time used.

13939 The **<sys/resource.h>** header shall define the **timeval** structure as described in **<sys/time.h>**.

13940 The **<sys/resource.h>** header shall define the following symbolic constants as possible values for
13941 the *resource* argument of *getrlimit*( ) and *setrlimit*( ):

13942 RLIMIT_CORE	Limit on size of core image.
13943 XSI RLIMIT_CPU	Limit on CPU time per process.

13944	RLIMIT_DATA	Limit on data segment size.
13945	RLIMIT_FSIZE	Limit on file size.
13946	RLIMIT_NOFILE	Limit on number of open files.
13947	RLIMIT_STACK	Limit on stack size.
13948	RLIMIT_AS	Limit on address space size.

13949 The following shall be declared as functions and may also be defined as macros. Function
13950 prototypes shall be provided.

```
13951 XSI  int  getpriority(int, id_t);
13952      int  getrlimit(int, struct rlimit *);
13953 XSI  int  getrusage(int, struct rusage *);
13954      int  setpriority(int, id_t, int);
13955      int  setrlimit(int, const struct rlimit *);
```

13956 XSI The **<sys/resource.h>** header shall define the **id_t** type through **typedef**, as described in
13957 **<sys/types.h>**.

13958 Inclusion of the **<sys/resource.h>** header may also make visible all symbols from **<sys/time.h>**.

13959 **APPLICATION USAGE**

13960 None.

13961 **RATIONALE**

13962 None.

13963 **FUTURE DIRECTIONS**

13964 None.

13965 **SEE ALSO**

13966 **<sys/time.h>**, **<sys/types.h>**

13967 XSH *getpriority*( ), *getrlimit*( ), *getrusage*( )

13968 **CHANGE HISTORY**

13969 First released in Issue 4, Version 2.

13970 **Issue 5**

13971 Large File System extensions are added.

13972 **Issue 7**

13973 This reference page is clarified with respect to macros and symbolic constants.

13974 **Issue 8**

13975 Austin Group Defects 51 and 1669 are applied, moving the *getrlimit*( ) and *setrlimit*( ) functions,
13976 excluding the RLIMIT_CPU limit, from the XSI option to the Base.

13977 Austin Group Defect 1141 is applied, changing the description of RLIMIT_CORE.

13978 **NAME**

13979 sys/select.h — select types

13980 **SYNOPSIS**

13981 `#include <sys/select.h>`

13982 **DESCRIPTION**

13983 The **<sys/select.h>** header shall define the **timeval** structure, which shall include at least the
13984 following members:

13985 `time_t` `tv_sec` Seconds.
13986 `suseconds_t` `tv_usec` Microseconds.

13987 The **<sys/select.h>** header shall define the **time_t** and **suseconds_t** types as described in
13988 **<sys/types.h>**.

13989 The **<sys/select.h>** header shall define the **sigset_t** type as described in **<signal.h>**.

13990 The **<sys/select.h>** header shall define the **timespec** structure as described in **<time.h>**.

13991 The **<sys/select.h>** header shall define the **fd_set** type as a structure.

13992 The **<sys/select.h>** header shall define the following symbolic constant, which shall have a value
13993 suitable for use in **#if** preprocessing directives:

13994 FD_SETSIZE Maximum number of file descriptors in an **fd_set** structure.

13995 The following shall be declared as functions, defined as macros, or both. If functions are
13996 declared, function prototypes shall be provided.

```
13997 void FD_CLR(int, fd_set *);
13998 int  FD_ISSET(int, const fd_set *);
13999 void FD_SET(int, fd_set *);
14000 void FD_ZERO(fd_set *);
```

14001 If implemented as macros, these may evaluate their arguments more than once, so applications
14002 should ensure that the arguments they supply are never expressions with side-effects.

14003 The following shall be declared as functions and may also be defined as macros. Function
14004 prototypes shall be provided.

```
14005 int  pselect(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
14006          const struct timespec *restrict, const sigset_t *restrict);
14007 int  select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
14008          struct timeval *restrict);
```

14009 Inclusion of the **<sys/select.h>** header may make visible all symbols from the headers
14010 **<signal.h>** and **<time.h>**.

14011 **APPLICATION USAGE**

14012 None.

14013 **RATIONALE**

14014 None.

14015 **FUTURE DIRECTIONS**

14016 None.

14017 **SEE ALSO**

14018 **<signal.h>**, **<sys/time.h>**, **<sys/types.h>**, **<time.h>**

14019 XSH *pselect*( )

14020 **CHANGE HISTORY**

14021 First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

14022 The requirement for the **fd_set** structure to have a member *fds_bits* has been removed as per The
14023 Open Group Base Resolution bwg2001-005.

14024 **Issue 7**

14025 SD5-XBD-ERN-6 is applied, reordering the DESCRIPTION.

14026 This reference page is clarified with respect to macros and symbolic constants.

14027 **Issue 8**

14028 Austin Group Defect 220 is applied, adding `const` to the second parameter of *FD_ISSET*( ).

14029 **NAME**

14030 sys/sem.h — XSI semaphore facility

14031 **SYNOPSIS**

14032 XSI
```
#include <sys/sem.h>
```

14033 **DESCRIPTION**

14034 The **<sys/sem.h>** header shall define the following symbolic constant for use as a semaphore
14035 operation flag:

14036 SEM_UNDO Set up adjust on exit entry.

14037 The **<sys/sem.h>** header shall define the following symbolic constants for use as commands for
14038 the *semctl*() function:

14039 GETNCNT Get *semncnt*.

14040 GETPID Get *sempid*.

14041 GETVAL Get *semval*.

14042 GETALL Get all cases of *semval*.

14043 GETZCNT Get *semzcnt*.

14044 SETVAL Set *semval*.

14045 SETALL Set all cases of *semval*.

14046 The **<sys/sem.h>** header shall define the **semid_ds** structure, which shall include the following
14047 members:

14048 `struct ipc_perm` `sem_perm` Operation permission structure.
14049 `unsigned short` `sem_nsems` Number of semaphores in set.
14050 `time_t` `sem_otime` Last *semop*() time.
14051 `time_t` `sem_ctime` Last time changed by *semctl*().

14052 The **<sys/sem.h>** header shall define the **pid_t**, **size_t**, and **time_t** types as described in
14053 **<sys/types.h>**.

14054 A semaphore shall be represented by an anonymous structure, which shall include the following
14055 members:

14056 `unsigned short` `semval` Semaphore value.
14057 `pid_t` `sempid` Process ID of last operation.
14058 `unsigned short` `semncnt` Number of processes waiting for *semval*
14059 to become greater than current value.
14060 `unsigned short` `semzcnt` Number of processes waiting for *semval*
14061 to become 0.

14062 The **<sys/sem.h>** header shall define the **sembuf** structure, which shall include the following
14063 members:

14064 `unsigned short` `sem_num` Semaphore number.
14065 `short` `sem_op` Semaphore operation.
14066 `short` `sem_flg` Operation flags.

14067 The following shall be declared as functions and may also be defined as macros. Function
14068 prototypes shall be provided.

14069
```
int   semctl(int, int, int, ...);
```



```
int   semget(key_t, int, int);
int   semop(int, struct sembuf *, size_t);
```

In addition, the **<sys/sem.h>** header shall include the **<sys/ipc.h>** header.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/ipc.h>, **<sys/types.h>**

XSH *semctl*( ), *semget*( ), *semop*( )

CHANGE HISTORY

First released in Issue 2. Derived from System V Release 2.0.

Issue 7

Austin Group Interpretation 1003.1-2001 #179 is applied.

This reference page is clarified with respect to macros and symbolic constants.

NAME

sys/shm.h — XSI shared memory facility

SYNOPSIS

XSI
```
#include <sys/shm.h>
```

DESCRIPTION

The **<sys/shm.h>** header shall define the following symbolic constants:

SHM_RDONLY Attach read-only (else read-write).

SHM_RND Round attach address to SHMLBA.

SHMLBA Segment low boundary address multiple.

The **<sys/shm.h>** header shall define the symbolic constant SHM_FAILED which shall evaluate to the same value as ((**void** *)(**intptr_t**)–1).

The **<sys/shm.h>** header shall define the type **intptr_t** as described in **<stdint.h>**.

The **<sys/shm.h>** header shall define the following data type through **typedef**:

shmatt_t Unsigned integer used for the number of current attaches that shall be able to store values at least as large as a type **unsigned short**.

The **<sys/shm.h>** header shall define the **shmid_ds** structure, which shall include the following members:

`struct ipc_perm`	`shm_perm`	Operation permission structure.
`size_t`	`shm_segsz`	Size of segment in bytes.
`pid_t`	`shm_lpid`	Process ID of last shared memory operation.
`pid_t`	`shm_cpid`	Process ID of creator.
`shmatt_t`	`shm_nattch`	Number of current attaches.
`time_t`	`shm_atime`	Time of last *shmat*( ).
`time_t`	`shm_dtime`	Time of last *shmdt*( ).
`time_t`	`shm_ctime`	Time of last change by *shmctl*( ).

The **<sys/shm.h>** header shall define the **pid_t**, **size_t**, and **time_t** types as described in **<sys/types.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
void *shmat(int, const void *, int);
int   shmctl(int, int, struct shmid_ds *);
int   shmdt(const void *);
int   shmget(key_t, size_t, int);
```

In addition, the **<sys/shm.h>** header shall include the **<sys/ipc.h>** header.

14121 **APPLICATION USAGE**

14122 None.

14123 **RATIONALE**

14124 None.

14125 **FUTURE DIRECTIONS**

14126 None.

14127 **SEE ALSO**

14128 **<sys/ipc.h>**, **<sys/types.h>**

14129 XSH *shmat*( ), *shmctl*( ), *shmdt*( ), *shmget*( )

14130 **CHANGE HISTORY**

14131 First released in Issue 2. Derived from System V Release 2.0.

14132 **Issue 5**

14133 The type of *shm_segsz* is changed from **int** to **size_t**.

14134 **Issue 7**

14135 Austin Group Interpretation 1003.1-2001 #179 is applied.

14136 This reference page is clarified with respect to macros and symbolic constants.

14137 **Issue 8**

14138 Austin Group Defect 1239 is applied, adding SHM_FAILED and requiring **<sys/shm.h>** to
14139 define **intptr_t**.

14140 **NAME**

14141 sys/socket.h — main sockets header

14142 **SYNOPSIS**

14143 `#include <sys/socket.h>`

14144 **DESCRIPTION**

14145 The **<sys/socket.h>** header shall define the **socklen_t** type, which is an integer type of width of
14146 at least 32 bits; see APPLICATION USAGE.

14147 The **<sys/socket.h>** header shall define the **sa_family_t** unsigned integer type.

14148 The **<sys/socket.h>** header shall define the **sockaddr** structure, which shall include at least the
14149 following members:

14150 `sa_family_t  sa_family` Address family.
14151 `char  sa_data[]` Socket address (variable-length data).

14152 The **sockaddr** structure is used to define a socket address which is used in the *bind*( ), *connect*( ),
14153 *getpeername*( ), *getsockname*( ), *recvfrom*( ), and *sendto*( ) functions.

14154 The **<sys/socket.h>** header shall define the **sockaddr_storage** structure, which shall be:

14155 • Large enough to accommodate all supported protocol-specific address structures

14156 • Aligned at an appropriate boundary so that pointers to it can be cast as pointers to
14157 protocol-specific address structures and used to access the fields of those structures
14158 without alignment problems

14159 The **sockaddr_storage** structure shall include at least the following members:

14160 `sa_family_t  ss_family`

14161 When a pointer to a **sockaddr_storage** structure is converted to a pointer to a **sockaddr**
14162 structure, or vice versa, the *ss_family* member of the **sockaddr_storage** structure shall map onto
14163 the *sa_family* member of the **sockaddr** structure. When a pointer to a **sockaddr_storage** structure
14164 is converted to a pointer to a protocol-specific address structure, or vice versa, the *ss_family*
14165 member shall map onto a member of that structure that is of type **sa_family_t** that identifies the
14166 protocol's address family. When a pointer to a **sockaddr** structure is converted to a pointer to a
14167 protocol-specific address structure, or vice versa, the *sa_family* member shall map onto a member
14168 of that structure that is of type **sa_family_t** that identifies the protocol's address family.
14169 Additionally, the structures shall be defined in such a way that the compiler treats an access to
14170 the stored value of the **sa_family_t** member of any of these structures, via an lvalue expression
14171 whose type involves any other one of these structures, as permissible, despite the more
14172 restrictive expression rules on stored value access as stated in the ISO C standard. Similarly,
14173 when a pointer to a **sockaddr_storage** or **sockaddr** structure is converted to a pointer to a
14174 protocol-specific address structure, the compiler shall treat an access (using this converted
14175 pointer) to the stored value of any member of the protocol-specific structure as permissible. The
14176 application shall ensure that the protocol-specific address structure corresponds to the family
14177 indicated by the member with type **sa_family_t** of that structure and the pointed-to object has
14178 sufficient memory for addressing all members of the protocol-specific structure.

14179 The **<sys/socket.h>** header shall define the **msghdr** structure, which shall include at least the
14180 following members:

14181 `void  *msg_name` Optional address.
14182 `socklen_t  msg_namelen` Size of address.
14183 `struct iovec  *msg_iov` Scatter/gather array.
14184 `int  msg_iovlen` Members in *msg_iov*.

`void`	`*msg_control`	Ancillary data; see below.
`socklen_t`	`msg_controllen`	Ancillary data buffer *len*.
`int`	`msg_flags`	Flags on received message.

The **msghdr** structure is used to reduce the number of directly supplied parameters to the *recvmsg*( ) and *sendmsg*( ) functions. This structure is used as a *value-result* parameter in the *recvmsg*( ) function and *value* only for the *sendmsg*( ) function.

The **<sys/socket.h>** header shall define the **iovec** structure as described in **<sys/uio.h>**.

The **<sys/socket.h>** header shall define the **cmsghdr** structure, which shall include at least the following members:

`socklen_t`	`cmsg_len`	Data byte count, including the **cmsghdr**.
`int`	`cmsg_level`	Originating protocol.
`int`	`cmsg_type`	Protocol-specific type.

The **cmsghdr** structure is used for storage of ancillary data object information.

Ancillary data consists of a sequence of pairs, each consisting of a **cmsghdr** structure followed by a data array. The data array contains the ancillary data message, and the **cmsghdr** structure contains descriptive information that allows an application to correctly parse the data.

The values for *cmsg_level* shall be legal values for the *level* argument to the *getsockopt*( ) and *setsockopt*( ) functions. The system documentation shall specify the *cmsg_type* definitions for the supported protocols.

Ancillary data is also possible at the socket level. The **<sys/socket.h>** header shall define the following symbolic constant for use as the *cmsg_type* value when *cmsg_level* is SOL_SOCKET:

SCM_RIGHTS — Indicates that the data array contains the access rights to be sent or received.

The **<sys/socket.h>** header shall define the following macros to gain access to the data arrays in the ancillary data associated with a message header:

CMSG_DATA(*cmsg*)

If the argument is a pointer to a **cmsghdr** structure, this macro shall return an unsigned character pointer to the data array associated with the **cmsghdr** structure.

CMSG_NXTHDR(*mhdr,cmsg*)

If the first argument is a pointer to a **msghdr** structure and the second argument is a pointer to a **cmsghdr** structure in the ancillary data pointed to by the *msg_control* field of that **msghdr** structure, this macro shall return a pointer to the next **cmsghdr** structure, or a null pointer if the second argument points to the last **cmsghdr** and data array pair in the ancillary data. If the ancillary data contains another **cmsghdr** structure after this one but the *cmsg_len* value in that structure is such that the data array following that structure would extend beyond the end of the ancillary data, it is unspecified whether this macro returns a pointer to that **cmsghdr** structure or returns a null pointer.

If the first argument is a pointer to a **msghdr** structure and the second argument is a null pointer, this macro shall be equivalent to CMSG_FIRSTHDR(*mhdr*).

CMSG_FIRSTHDR(*mhdr*)

If the argument is a pointer to a **msghdr** structure, this macro shall return a pointer to the first **cmsghdr** structure in the ancillary data associated with this **msghdr** structure, or a null pointer if either there is no ancillary data associated with the **msghdr** structure (*msg_controllen* is zero) or there is insufficient room in the ancillary data for a complete **cmsghdr** structure (*msg_controllen* is non-zero but less than `sizeof(struct cmsghdr)`).

CMSG_SPACE(*length*)
If the argument has a type such that its value can be assigned to an object of type **socklen_t**, this macro shall return the space required by an ancillary data object of the specified length and its **cmsghdr** structure, including any padding needed to satisfy alignment requirements. This macro can be used, for example, to allocate space dynamically for the ancillary data. This macro should not be used to initialize the *cmsg_len* member of a **cmsghdr** structure. If the argument is an integer constant expression, this macro shall expand to an integer constant expression.

CMSG_LEN(*length*)
If the argument has a type such that its value can be assigned to an object of type **socklen_t**, this macro shall return the value to store in the *cmsg_len* member of the **cmsghdr** structure for an ancillary data object of the specified length, taking into account any padding needed to satisfy alignment requirements. If the argument is an integer constant expression, this macro shall expand to an integer constant expression.

The **<sys/socket.h>** header shall define the **linger** structure, which shall include at least the following members:

`int l_onoff` Indicates whether linger option is enabled.
`int l_linger` Linger time, in seconds.

The **<sys/socket.h>** header shall define the following socket types (see XSH Section 2.10.6, on page 550) as symbolic constants with distinct values:

SOCK_DGRAM Datagram socket.

RS SOCK_RAW Raw Protocol Interface.

SOCK_SEQPACKET Sequenced-packet socket.

SOCK_STREAM Byte-stream socket.

Implementations may provide additional socket types.

The header shall define the following socket creation flags, for use in *socket*( ), *socketpair*( ), and *accept4*( ). These flags shall be symbolic constants with values that are bitwise distinct from each other and from all SOCK_* constants representing socket types:

SOCK_NONBLOCK Create a socket file descriptor with the O_NONBLOCK flag atomically set on the new open file description.

SOCK_CLOEXEC Create a socket file descriptor with the FD_CLOEXEC flag atomically set on that file descriptor.

SOCK_CLOFORK Create a socket file descriptor with the FD_CLOFORK flag atomically set on that file descriptor.

Implementations may provide additional socket creation flags.

The **<sys/socket.h>** header shall define the following symbolic constant for use as the *level* argument of *setsockopt*( ) and *getsockopt*( ).

SOL_SOCKET Options to be accessed at socket level, not protocol level.

The **<sys/socket.h>** header shall define the following symbolic constants with distinct values for use as the *option_name* argument in *getsockopt*( ) or *setsockopt*( ) calls (see XSH Section 2.10.16, on page 554):

SO_ACCEPTCONN	Socket is accepting connections.
SO_BROADCAST	Transmission of broadcast messages is supported.
SO_DEBUG	Debugging information is being recorded.
SO_DOMAIN	Socket domain.
SO_DONTROUTE	Bypass normal routing.
SO_ERROR	Socket error status.
SO_KEEPALIVE	Connections are kept alive with periodic messages.
SO_LINGER	Socket lingers on close.
SO_OOBINLINE	Out-of-band data is transmitted in line.
SO_PROTOCOL	Socket protocol.
SO_RCVBUF	Receive buffer size.
SO_RCVLOWAT	Receive ``low water mark''.
SO_RCVTIMEO	Receive timeout.
SO_REUSEADDR	Reuse of local addresses is supported.
SO_SNDBUF	Send buffer size.
SO_SNDLOWAT	Send ``low water mark''.
SO_SNDTIMEO	Send timeout.
SO_TYPE	Socket type.

The **<sys/socket.h>** header shall define the following symbolic constant for use as the maximum *backlog* queue length which may be specified by the *backlog* field of the *listen*( ) function:

SOMAXCONN	The maximum *backlog* queue length.

The **<sys/socket.h>** header shall define the following symbolic constants with distinct values for use as the valid values for the *msg_flags* field in the **msghdr** structure, or the *flags* parameter in *recv*( ), *recvfrom*( ), *recvmsg*( ), *send*( ), *sendmsg*( ), or *sendto*( ) calls:

MSG_CMSG_CLOEXEC	Atomically set the FD_CLOEXEC flag on any file descriptors created via SCM_RIGHTS during *recvmsg*( ).
MSG_CMSG_CLOFORK	Atomically set the FD_CLOFORK flag on any file descriptors created via SCM_RIGHTS during *recvmsg*( ).
MSG_CTRUNC	Control data truncated.
MSG_DONTROUTE	Send without using routing tables.
MSG_EOR	Terminates a record (if supported by the protocol).
MSG_OOB	Out-of-band data.
MSG_NOSIGNAL	No SIGPIPE generated when an attempt to send is made on a stream-oriented socket that is no longer connected.

14307 MSG_PEEK Leave received data in queue.

14308 MSG_TRUNC Normal data truncated.

14309 MSG_WAITALL Attempt to fill the read buffer.

14310 The **<sys/socket.h>** header shall define the following symbolic constants with distinct values:

14311 AF_INET Internet domain sockets for use with IPv4 addresses.

14312 IP6 AF_INET6 Internet domain sockets for use with IPv6 addresses.

14313 AF_UNIX UNIX domain sockets.

14314 AF_UNSPEC Unspecified.

14315 The value of AF_UNSPEC shall be 0.

14316 The **<sys/socket.h>** header shall define the following symbolic constants with distinct values:

14317 SHUT_RD Disables further receive operations.

14318 SHUT_RDWR Disables further send and receive operations.

14319 SHUT_WR Disables further send operations.

14320 The **<sys/socket.h>** header shall define the **size_t** and **ssize_t** types as described in
14321 **<sys/types.h>**.

14322 The following shall be declared as functions and may also be defined as macros. Function
14323 prototypes shall be provided.

```
14324 int     accept(int, struct sockaddr *restrict, socklen_t *restrict);
14325 int     accept4(int, struct sockaddr *restrict, socklen_t *restrict,
14326         int);
14327 int     bind(int, const struct sockaddr *, socklen_t);
14328 int     connect(int, const struct sockaddr *, socklen_t);
14329 int     getpeername(int, struct sockaddr *restrict, socklen_t *restrict);
14330 int     getsockname(int, struct sockaddr *restrict, socklen_t *restrict);
14331 int     getsockopt(int, int, int, void *restrict, socklen_t *restrict);
14332 int     listen(int, int);
14333 ssize_t recv(int, void *, size_t, int);
14334 ssize_t recvfrom(int, void *restrict, size_t, int,
14335         struct sockaddr *restrict, socklen_t *restrict);
14336 ssize_t recvmsg(int, struct msghdr *, int);
14337 ssize_t send(int, const void *, size_t, int);
14338 ssize_t sendmsg(int, const struct msghdr *, int);
14339 ssize_t sendto(int, const void *, size_t, int, const struct sockaddr *,
14340         socklen_t);
14341 int     setsockopt(int, int, int, const void *, socklen_t);
14342 int     shutdown(int, int);
14343 int     sockatmark(int);
14344 int     socket(int, int, int);
14345 int     socketpair(int, int, int, int [2]);
```

14346 Inclusion of **<sys/socket.h>** may also make visible all symbols from **<sys/uio.h>**.

APPLICATION USAGE

To forestall portability problems, it is recommended that applications not use values larger than 2^{31} –1 for the **socklen_t** type.

The **sockaddr_storage** structure solves the problem of declaring storage for automatic variables which is both large enough and aligned enough for storing the socket address data structure of any family. For example, code with a file descriptor and without the context of the address family can pass a pointer to a variable of this type, where a pointer to a socket address structure is expected in calls such as *getpeername*( ), and determine the address family by accessing the received content after the call.

The example below illustrates a data structure which aligns on a 64-bit boundary. An implementation-defined field *_ss_align* following *_ss_pad1* is used to force a 64-bit alignment which covers proper alignment good enough for needs of at least **sockaddr_in6** (IPv6) and **sockaddr_in** (IPv4) address data structures. The size of padding field *_ss_pad1* depends on the chosen alignment boundary. The size of padding field *_ss_pad2* depends on the value of overall size chosen for the total size of the structure. This size and alignment are represented in the above example by implementation-defined (not required) constants _SS_MAXSIZE (chosen value 128) and _SS_ALIGNMENT (with chosen value 8). Constants _SS_PAD1SIZE (derived value 6) and _SS_PAD2SIZE (derived value 112) are also for illustration and not required. The implementation-defined definitions and structure field names above start with an <underscore> to denote implementation private name space. Portable code is not expected to access or reference those fields or constants. Note that this example only deals with size and alignment; see RATIONALE for additional issues related to these structures.

```
/*
 *  Desired design of maximum size and alignment.
 */
#define _SS_MAXSIZE 128
    /* Implementation-defined maximum size. */
#define _SS_ALIGNSIZE (sizeof(int64_t))
    /* Implementation-defined desired alignment. */

/*
 *  Definitions used for sockaddr_storage structure paddings design.
 */
#define _SS_PAD1SIZE (_SS_ALIGNSIZE - sizeof(sa_family_t))
#define _SS_PAD2SIZE (_SS_MAXSIZE - (sizeof(sa_family_t)+ \
                      _SS_PAD1SIZE + _SS_ALIGNSIZE))
struct sockaddr_storage {
    sa_family_t  ss_family;  /* Address family. */
/*
 *  Following fields are implementation-defined.
 */
    char _ss_pad1[_SS_PAD1SIZE];
        /* 6-byte pad; this is to make implementation-defined
           pad up to alignment field that follows explicit in
           the data structure. */
    int64_t _ss_align;  /* Field to force desired structure
                           storage alignment. */
    char _ss_pad2[_SS_PAD2SIZE];
        /* 112-byte pad to achieve desired size,
           _SS_MAXSIZE value minus size of ss_family
           __ss_pad1, __ss_align fields is 112. */
```



```
};
```

Portable applications need to account for the alternative behaviors of the CMSG_NXTHDR macro as follows:

- When constructing ancillary data in a **msghdr** structure, ensure that all locations within the ancillary data that might be returned by CMSG_NXTHDR contain a *cmsg_len* value of zero (typically this is achieved by using *memset*( ) to initialize the entire *msg_control* buffer to null bytes before populating the first **cmsghdr** structure).
- When extracting ancillary data from a received **msghdr** structure, check that the data array following the last **cmsghdr** structure does not extend beyond the end of the ancillary data.

RATIONALE

Note that defining the **sockaddr_storage** and **sockaddr** structures using only mechanisms defined in early editions of the ISO C standard may produce aliasing diagnostics when applications use casting between pointers to the various socket address structures. Because of the large body of existing code utilizing sockets in a way that could trigger undefined behavior due to strict aliasing rules, this standard mandates that these structures can alias each other for accessing the **sa_family_t** member of the structures (or other members for protocol-specific structure references), so as to preserve well-defined semantics. An implementation's header files may need to use anonymous unions, or even an implementation-specific extension, to comply with the requirements of this standard.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, **<sys/uio.h>**

XSH *accept*( ), *bind*( ), *connect*( ), *getpeername*( ), *getsockname*( ), *getsockopt*( ), *listen*( ), *recv*( ), *recvfrom*( ), *recvmsg*( ), *send*( ), *sendmsg*( ), *sendto*( ), *setsockopt*( ), *shutdown*( ), *sockatmark*( ), *socket*( ), *socketpair*( )

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The **restrict** keyword is added to the prototypes for *accept*( ), *getpeername*( ), *getsockname*( ), *getsockopt*( ), and *recvfrom*( ).

Issue 7

SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **ssize_t** type.

SD5-XBD-ERN-62 is applied.

The MSG_NOSIGNAL symbolic constant is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

This reference page is clarified with respect to macros and symbolic constants, and a declaration for the **size_t** type is added.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0067 [355] is applied.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0077 [934] is applied.

Issue 8

Austin Group Defects 411 and 1318 are applied, adding SOCK_NONBLOCK, SOCK_CLOEXEC, SOCK_CLOFORK, MSG_CMSG_CLOEXEC, MSG_CMSG_CLOFORK, and *accept4*( ).

Austin Group Defect 840 is applied, adding SO_DOMAIN and SO_PROTOCOL.

Austin Group Defect 978 is applied, adding CMSG_SPACE and CMSG_LEN, and clarifying the behavior of CMSG_NXTHDR when the second argument is a null pointer.

Austin Group Defect 1056 is applied, clarifying the conditions under which CMSG_NXTHDR and CMSG_FIRSTHDR return a null pointer, and adding a new paragraph to APPLICATION USAGE.

Austin Group Defect 1641 is applied, clarifying the requirements for conversions between pointers to **sockaddr_storage** and **storage** structures and between pointers to those structures and pointers to protocol-specific address structures.

14448 **NAME**

14449 sys/stat.h — data returned by the stat( ) function

14450 **SYNOPSIS**

14451 `#include <sys/stat.h>`

14452 **DESCRIPTION**

14453 The **<sys/stat.h>** header shall define the structure of the data returned by the *fstat*( ), *lstat*( ), and
14454 *stat*( ) functions.

14455 The **<sys/stat.h>** header shall define the **stat** structure, which shall include at least the following
14456 members:

14457	`dev_t st_dev`	Device ID of device containing file.
14458	`ino_t st_ino`	File serial number.
14459	`mode_t st_mode`	Mode of file (see below).
14460	`nlink_t st_nlink`	Number of hard links to the file.
14461	`uid_t st_uid`	User ID of file.
14462	`gid_t st_gid`	Group ID of file.
14463 XSI	`dev_t st_rdev`	Device ID (if file is character or block special).
14464	`off_t st_size`	For regular files, the file size in bytes.
14465		For symbolic links, the length in bytes of the
14466		pathname contained in the symbolic link.
14467 SHM		For a shared memory object, the length in bytes.
14468 TYM		For a typed memory object, the length in bytes.
14469		For other file types, the use of this field is
14470		unspecified.
14471	`struct timespec st_atim`	Last data access timestamp.
14472	`struct timespec st_mtim`	Last data modification timestamp.
14473	`struct timespec st_ctim`	Last file status change timestamp.
14474 XSI	`blksize_t st_blksize`	A file system-specific preferred I/O block size
14475		for this object. In some file system types, this
14476		may vary from file to file.
14477	`blkcnt_t st_blocks`	Number of blocks allocated for this object.

14478 A *file identity* is uniquely determined by the combination of *st_dev* and *st_ino*. At any given time
14479 in a system, distinct files shall have distinct file identities; hard links to the same file shall have
14480 the same file identity. Over time, these file identities can be reused for different files. For
14481 example, the *st_ino* value can be reused after the last link to a file is unlinked and the space
14482 occupied by the file has been freed, and the *st_dev* value associated with a file system can be
14483 reused if that file system is detached (``unmounted'') and another is attached (``mounted'').

14484 The *st_nlink* value shall be the number of hard links to the file within the file system in which the
14485 file resides.

14486 **Note:** The number of links to the file that can be found by traversing the file hierarchy can differ from
14487 *st_nlink*. For example, it can be less than *st_nlink* if a link to the file cannot be reached because it
14488 is below a directory that has been overlaid with a mount point for a different file system, and it
14489 can be greater than *st_nlink* on implementations that allow a file system (or part of one) to be
14490 duplicated at additional mount points.

14491 XSI The **<sys/stat.h>** header shall define the **blkcnt_t**, **blksize_t**, **dev_t**, **ino_t**, **mode_t**, **nlink_t**,
14492 **uid_t**, **gid_t**, **off_t**, and **time_t** types as described in **<sys/types.h>**.

14493 The **<sys/stat.h>** header shall define the **timespec** structure as described in **<time.h>**. Times
14494 shall be given in seconds since the Epoch.

Which structure members have meaningful values depends on the type of file. For further information, see the descriptions of *fstat*( ), *lstat*( ), and *stat*( ) in the System Interfaces volume of POSIX.1-2024.

For compatibility with earlier versions of this standard, the *st_atime* macro shall be defined with the value *st_atim.tv_sec*. Similarly, *st_ctime* and *st_mtime* shall be defined as macros with the values *st_ctim.tv_sec* and *st_mtim.tv_sec*, respectively.

The **<sys/stat.h>** header shall define the following symbolic constants for the file types encoded in type **mode_t**. The values shall be suitable for use in **#if** preprocessing directives:

XSI

S_IFMT	Type of file.
S_IFBLK	Block special.
S_IFCHR	Character special.
S_IFIFO	FIFO special.
S_IFREG	Regular.
S_IFDIR	Directory.
S_IFLNK	Symbolic link.
S_IFSOCK	Socket.

The **<sys/stat.h>** header shall define the following symbolic constants for the file mode bits encoded in type **mode_t**, with the indicated numeric values. These macros shall expand to an expression which has a type that allows them to be used, either singly or OR'ed together, as the third argument to *open*( ) without the need for a **mode_t** cast. The values shall be suitable for use in **#if** preprocessing directives.

Name	Numeric Value	Description
S_IRWXU	0700	Read, write, execute/search by owner.
S_IRUSR	0400	Read permission, owner.
S_IWUSR	0200	Write permission, owner.
S_IXUSR	0100	Execute/search permission, owner.
S_IRWXG	070	Read, write, execute/search by group.
S_IRGRP	040	Read permission, group.
S_IWGRP	020	Write permission, group.
S_IXGRP	010	Execute/search permission, group.
S_IRWXO	07	Read, write, execute/search by others.
S_IROTH	04	Read permission, others.
S_IWOTH	02	Write permission, others.
S_IXOTH	01	Execute/search permission, others.
S_ISUID	04000	Set-user-ID on execution.
S_ISGID	02000	Set-group-ID on execution.
S_ISVTX	01000	On directories, restricted deletion flag.

XSI

The following macros shall be provided to test whether a file is of the specified type. The value *m* supplied to the macros is the value of *st_mode* from a **stat** structure. The macro shall evaluate to a non-zero value if the test is true; 0 if the test is false.

S_ISBLK(*m*) Test for a block special file.

14536 S_ISCHR(*m*) Test for a character special file.

14537 S_ISDIR(*m*) Test for a directory.

14538 S_ISFIFO(*m*) Test for a pipe or FIFO special file.

14539 S_ISREG(*m*) Test for a regular file.

14540 S_ISLNK(*m*) Test for a symbolic link.

14541 S_ISSOCK(*m*) Test for a socket.

14542 The implementation may implement message queues, semaphores, or shared memory objects as
14543 distinct file types, in which case these file types need not be encoded in type **mode_t**. The
14544 following macros shall be provided to test whether a file is of the specified type. The value of the
14545 *buf* argument supplied to the macros is a pointer to a **stat** structure. The macro shall evaluate to a
14546 non-zero value if the specified object is implemented as a distinct file type and the specified file
14547 type is contained in the **stat** structure referenced by *buf*. Otherwise, the macro shall evaluate to
14548 zero.

14549 S_TYPEISMQ(*buf*) Test for a message queue.

14550 S_TYPEISSEM(*buf*) Test for a semaphore.

14551 S_TYPEISSHM(*buf*) Test for a shared memory object.

14552 TYM The implementation may implement typed memory objects as a distinct file type, in which case
14553 this file type need not be encoded in type **mode_t**. The following macro shall test whether a file
14554 is of the specified type. The value of the *buf* argument supplied to the macros is a pointer to a
14555 **stat** structure. The macro shall evaluate to a non-zero value if the specified object is implemented
14556 as a distinct file type and the specified file type is contained in the **stat** structure referenced by
14557 *buf*. Otherwise, the macro shall evaluate to zero.

14558 S_TYPEISTMO(*buf*) Test macro for a typed memory object.

14559 The **<sys/stat.h>** header shall define the following symbolic constants as distinct integer values
14560 outside of the range [0,999 999 999], for use with the *futimens*( ) and *utimensat*( ) functions:

14561 UTIME_NOW
14562 UTIME_OMIT

14563 The following shall be declared as functions and may also be defined as macros. Function
14564 prototypes shall be provided.

```
14565 int    chmod(const char *, mode_t);
14566 int    fchmod(int, mode_t);
14567 int    fchmodat(int, const char *, mode_t, int);
14568 int    fstat(int, struct stat *);
14569 int    fstatat(int, const char *restrict, struct stat *restrict, int);
14570 int    futimens(int, const struct timespec [2]);
14571 int    lstat(const char *restrict, struct stat *restrict);
14572 int    mkdir(const char *, mode_t);
14573 int    mkdirat(int, const char *, mode_t);
14574 int    mkfifo(const char *, mode_t);
14575 int    mkfifoat(int, const char *, mode_t);
14576 XSI    int    mknod(const char *, mode_t, dev_t);
14577        int    mknodat(int, const char *, mode_t, dev_t);
14578 int    stat(const char *restrict, struct stat *restrict);
```



```
mode_t umask(mode_t);
int    utimensat(int, const char *, const struct timespec [2], int);
```

Inclusion of the **<sys/stat.h>** header may make visible all symbols from the **<time.h>** header.

APPLICATION USAGE

Use of the macros is recommended for determining the type of a file.

RATIONALE

A conforming C-language application must include **<sys/stat.h>** for functions that have arguments or return values of type **mode_t**, so that symbolic values for that type can be used. An alternative would be to require that these constants are also defined by including **<sys/types.h>**.

The S_ISUID and S_ISGID bits may be cleared on any write, not just on *open*(), as some historical implementations do.

System calls that update the time entry fields in the **stat** structure must be documented by the implementors. POSIX-conforming systems should not update the time entry fields for functions listed in the System Interfaces volume of POSIX.1-2024 unless the standard requires that they do, except in the case of documented extensions to the standard.

Upon assignment, file timestamps are immediately converted to the resolution of the file system by truncation (i.e., the recorded time can be older than the actual time). For example, if the file system resolution is 1 microsecond, then a conforming *stat*() must always return an *st_mtim.tv_nsec* that is a multiple of 1000. Some older implementations returned higher-resolution timestamps while the *inode* information was cached, and then spontaneously truncated the *tv_nsec* fields when they were stored to and retrieved from disk, but this behavior does not conform.

Note that *st_dev* must be unique within a Local Area Network (LAN) in a ``system'' made up of multiple computers' file systems connected by a LAN.

Networked implementations of a POSIX-conforming system must guarantee that all files visible within the file tree (including parts of the tree that may be remotely mounted from other machines on the network) on each individual processor are uniquely identified by the combination of the *st_ino* and *st_dev* fields.

The unit for the *st_blocks* member of the **stat** structure is not defined within POSIX.1-2024. In some implementations it is 512 bytes. It may differ on a file system basis. There is no correlation between values of the *st_blocks* and *st_blksize*, and the *f_bsize* (from **<sys/statvfs.h>**) structure members.

Traditionally, some implementations defined the multiplier for *st_blocks* in **<sys/param.h>** as the symbol DEV_BSIZE.

Some earlier versions of this standard did not specify values for the file mode bit macros. The expectation was that some implementors might choose to use a different encoding for these bits than the traditional one, and that new applications would use symbolic file modes instead of numeric. This version of the standard specifies the traditional encoding, in recognition that nearly 20 years after the first publication of this standard numeric file modes are still in widespread use by application developers, and that all conforming implementations still use the traditional encoding.

FUTURE DIRECTIONS

No new S_IFMT symbolic names for the file type values of **mode_t** will be defined by POSIX.1-2024; if new file types are required, they will only be testable through *S_ISxx*( ) or *S_TYPEISxxx*( ) macros instead.

SEE ALSO

<sys/statvfs.h>, **<sys/types.h>**, **<time.h>**

XSH *chmod*( ), *fchmod*( ), *fstat*( ), *fstatat*( ), *futimens*( ), *mkdir*( ), *mkfifo*( ), *mknod*( ), *umask*( )

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

The type of *st_blksize* is changed from **long** to **blksize_t**; the type of *st_blocks* is changed from **long** to **blkcnt_t**.

Issue 6

The S_TYPEISMQ( ), S_TYPEISSEM( ), and S_TYPEISSHM( ) macros are unconditionally mandated.

The Open Group Corrigendum U035/4 is applied. In the DESCRIPTION, the types **blksize_t** and **blkcnt_t** have been described.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **dev_t**, **ino_t**, **mode_t**, **nlink_t**, **uid_t**, **gid_t**, **off_t**, and **time_t** types are mandated.

S_IFSOCK and S_ISSOCK are added for sockets.

The description of **stat** structure members is changed to reflect contents when file type is a symbolic link.

The test macro S_TYPEISTMO is added for alignment with IEEE Std 1003.1j-2000.

The **restrict** keyword is added to the prototypes for *lstat*( ) and *stat*( ).

The *lstat*( ) function is made mandatory.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/17 is applied, adding text regarding the *st_blocks* member of the **stat** structure to the RATIONALE.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/25 is applied, adding to the DESCRIPTION that the **timespec** structure may be defined as described in the **<time.h>** header.

Issue 7

SD5-XSH-ERN-161 is applied, updating the DESCRIPTION to clarify that the descriptions of the interfaces should be consulted in order to determine which structure members have meaningful values.

The *fchmodat*( ), *fstatat*( ), *mkdirat*( ), *mkfifoat*( ), *mknodat*( ), and *utimensat*( ) functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The *futimens*( ) function is added.

This reference page is clarified with respect to macros and symbolic constants.

Changes are made related to support for finegrained timestamps and the UTIME_NOW and UTIME_OMIT symbolic constants are added.

14662 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0068 [207] is applied.

14663 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0078 [531] is applied.

14664 **Issue 8**

14665 Austin Group Defect 732 is applied, clarifying that if message queues, semaphores, shared
14666 memory objects, or typed memory objects are implemented as distinct file types, they need not
14667 be encoded in type **mode_t**.

14668 Austin Group Defect 1314 is applied, clarifying how the *st_dev* and *st_ino* values identify files.

14669 Austin Group Defect 1323 is applied, clarifying how the *st_nlink* value relates to the number of
14670 links to the file that can be found by traversing the file hierarchy.

NAME

sys/statvfs.h — VFS File System information structure

SYNOPSIS

```
#include <sys/statvfs.h>
```

DESCRIPTION

The **<sys/statvfs.h>** header shall define the **statvfs** structure, which shall include at least the following members:

`unsigned long`	`f_bsize`	File system block size.
`unsigned long`	`f_frsize`	Fundamental file system block size.
`fsblkcnt_t`	`f_blocks`	Total number of blocks on file system in units of *f_frsize.*
`fsblkcnt_t`	`f_bfree`	Total number of free blocks.
`fsblkcnt_t`	`f_bavail`	Number of free blocks available to non-privileged process.
`fsfilcnt_t`	`f_files`	Total number of file serial numbers.
`fsfilcnt_t`	`f_ffree`	Total number of free file serial numbers.
`fsfilcnt_t`	`f_favail`	Number of file serial numbers available to non-privileged process.
`unsigned long`	`f_fsid`	File system ID.
`unsigned long`	`f_flag`	Bit mask of *f_flag* values.
`unsigned long`	`f_namemax`	Maximum filename length.

The **<sys/statvfs.h>** header shall define the **fsblkcnt_t** and **fsfilcnt_t** types as described in **<sys/types.h>**.

The **<sys/statvfs.h>** header shall define the following symbolic constants for the *f_flag* member:

ST_RDONLY Read-only file system.

ST_NOSUID Does not support the semantics of the ST_ISUID and ST_ISGID file mode bits.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int fstatvfs(int, struct statvfs *);
int statvfs(const char *restrict, struct statvfs *restrict);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>

XSH *fstatvfs*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

The type of *f_blocks*, *f_bfree*, and *f_bavail* is changed from **unsigned long** to **fsblkcnt_t**; the type of *f_files*, *f_ffree*, and *f_favail* is changed from **unsigned long** to **fsfilcnt_t**.

Issue 6

The Open Group Corrigendum U035/5 is applied. In the DESCRIPTION, the types **fsblkcnt_t** and **fsfilcnt_t** have been described.

The **restrict** keyword is added to the prototype for *statvfs*().

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/18 is applied, changing the description of ST_NOSUID from ``Does not support *setuid*()/*setgid*() semantics'' to ``Does not support the semantics of the ST_ISUID and ST_ISGID file mode bits''.

Issue 7

The **<sys/statvfs.h>** header is moved from the XSI option to the Base.

This reference page is clarified with respect to macros and symbolic constants.

NAME

sys/time.h — time types

SYNOPSIS

XSI
```
#include <sys/time.h>
```

DESCRIPTION

The **<sys/time.h>** header shall define the **fd_set** type and the **timeval** structure, as described in **<sys/select.h>**.

The **<sys/time.h>** header shall define the **time_t** and **suseconds_t** types as described in **<sys/types.h>**.

The **<sys/time.h>** header shall define the following as described in **<sys/select.h>**:

FD_CLR( )
FD_ISSET( )
FD_SET( )
FD_ZERO( )
FD_SETSIZE

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int   select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
          struct timeval *restrict);
int   utimes(const char *, const struct timeval [2]);
```

Inclusion of the **<sys/time.h>** header may make visible all symbols from the **<sys/select.h>** header.

APPLICATION USAGE

None.

RATIONALE

The **<sys/time.h>** header refers to **<sys/select.h>** for the definition of the **timeval** structure, instead of the other way round, because **<sys/time.h>** is an optional (XSI) header whereas **<sys/select.h>** is mandatory.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/select.h>, **<sys/types.h>**

XSH *futimens*( ), *pselect*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

The type of *tv_usec* is changed from **long** to **suseconds_t**.

Issue 6

The **restrict** keyword is added to the prototypes for *gettimeofday*( ), *select*( ), and *setitimer*( ).

The note is added that inclusion of this header may also make symbols visible from **<sys/select.h>**.

The *utimes*( ) function is marked LEGACY.

14766 **Issue 7**

14767 This reference page is clarified with respect to macros and symbolic constants.

14768 **Issue 8**

14769 Austin Group Defect 1171 is applied, replacing the **timeval** structure definition with a reference
14770 to its description in **<sys/select.h>**.

14771 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

NAME

sys/times.h — file access and modification times structure

SYNOPSIS

```
#include <sys/times.h>
```

DESCRIPTION

The **<sys/times.h>** header shall define the **tms** structure, which is returned by *times*() and shall include at least the following members:

```
clock_t  tms_utime   User CPU time.
clock_t  tms_stime   System CPU time.
clock_t  tms_cutime  User CPU time of terminated child processes.
clock_t  tms_cstime  System CPU time of terminated child processes.
```

The **<sys/times.h>** header shall define the **clock_t** type as described in **<sys/types.h>**.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
clock_t times(struct tms *);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>

XSH *times*()

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

14798 **NAME**

14799 sys/types.h — data types

14800 **SYNOPSIS**

14801 `#include <sys/types.h>`

14802 **DESCRIPTION**

14803 The **<sys/types.h>** header shall define at least the following types:

14804	**blkcnt_t**	Used for file block counts.
14805	**blksize_t**	Used for block sizes.
14806 14807	**clock_t**	Used for system times in clock ticks or CLOCKS_PER_SEC; see **<time.h>**.
14808	**clockid_t**	Used for clock ID type in the clock and timer functions.
14809	**dev_t**	Used for device IDs.
14810	**fsblkcnt_t**	Used for file system block counts.
14811	**fsfilcnt_t**	Used for file system file counts.
14812	**gid_t**	Used for group IDs.
14813 14814	**id_t**	Used as a general identifier; can be used to contain at least a **pid_t**, **uid_t**, or **gid_t**.
14815	**ino_t**	Used for file serial numbers.
14816 XSI	**key_t**	Used for XSI interprocess communication.
14817	**mode_t**	Used for some file attributes.
14818	**nlink_t**	Used for link counts.
14819	**off_t**	Used for file sizes.
14820	**pid_t**	Used for process IDs and process group IDs.
14821	**pthread_attr_t**	Used to identify a thread attribute object.
14822	**pthread_barrier_t**	Used to identify a barrier.
14823	**pthread_barrierattr_t**	Used to define a barrier attributes object.
14824	**pthread_cond_t**	Used for condition variables.
14825	**pthread_condattr_t**	Used to identify a condition attribute object.
14826	**pthread_key_t**	Used for thread-specific data keys.
14827	**pthread_mutex_t**	Used for mutexes.
14828	**pthread_mutexattr_t**	Used to identify a mutex attribute object.
14829	**pthread_once_t**	Used for dynamic package initialization.
14830	**pthread_rwlock_t**	Used for read-write locks.
14831	**pthread_rwlockattr_t**	Used for read-write lock attributes.
14832	**pthread_spinlock_t**	Used to identify a spin lock.

pthread_t	Used to identify a thread.
reclen_t	Used for directory entry lengths.
size_t	Used for sizes of objects.
ssize_t	Used for a count of bytes or an error indication.
suseconds_t	Used for time in microseconds.
time_t	Used for time in seconds.
timer_t	Used for timer ID returned by *timer_create*( ).
uid_t	Used for user IDs.

All of the types shall be defined as arithmetic types of an appropriate length, with the following exceptions:

pthread_attr_t
pthread_barrier_t
pthread_barrierattr_t
pthread_cond_t
pthread_condattr_t
pthread_key_t
pthread_mutex_t
pthread_mutexattr_t
pthread_once_t
pthread_rwlock_t
pthread_rwlockattr_t
pthread_spinlock_t
pthread_t
timer_t

Additionally:

- **mode_t** shall be an integer type.
- **dev_t** shall be an integer type.
- **nlink_t**, **uid_t**, **gid_t**, and **id_t** shall be integer types.
- **blkcnt_t** and **off_t** shall be signed integer types.
- **fsblkcnt_t**, **fsfilcnt_t**, **reclen_t**, and **ino_t** shall be defined as unsigned integer types.
- **size_t** shall be an unsigned integer type.
- **blksize_t**, **pid_t**, and **ssize_t** shall be signed integer types.
- **clock_t** shall be an integer or real-floating type.
- CX **time_t** shall be an integer type with a width (see **<stdint.h>**) of at least 64 bits.

The type **ssize_t** shall be capable of storing values at least in the range [–1, {SSIZE_MAX}].

XSI The type **suseconds_t** shall be a signed integer type capable of storing values at least in the range [–1, 1 000 000].

The implementation shall support one or more programming environments in which the widths of **blksize_t**, **pid_t**, **size_t**, **ssize_t**, and **suseconds_t** are no greater than the width of type **long**. The names of these programming environments can be obtained using the *confstr*( ) function or the *getconf* utility.

There are no defined comparison or assignment operators for the following types:

pthread_attr_t
pthread_barrier_t
pthread_barrierattr_t
pthread_cond_t
pthread_condattr_t
pthread_mutex_t
pthread_mutexattr_t
pthread_rwlock_t
pthread_rwlockattr_t
pthread_spinlock_t
timer_t

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<stdint.h>, **<time.h>**

XSH *confstr( )*

XCU *getconf*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The **clockid_t** and **timer_t** types are defined for alignment with the POSIX Realtime Extension.

The types **blkcnt_t**, **blksize_t**, **fsblkcnt_t**, **fsfilcnt_t**, and **suseconds_t** are added.

Large File System extensions are added.

Updated for alignment with the POSIX Threads Extension.

Issue 6

The **pthread_barrier_t**, **pthread_barrierattr_t**, and **pthread_spinlock_t** types are added for alignment with IEEE Std 1003.1j-2000.

The margin code is changed from XSI to THR for the **pthread_rwlock_t** and **pthread_rwlockattr_t** types as Read-Write Locks have been absorbed into the POSIX Threads option. The threads types are marked THR.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/26 is applied, adding **pthread_t** to the list of types that are not required to be arithmetic types, thus allowing **pthread_t** to be defined as a structure.

Issue 7

Austin Group Interpretation 1003.1-2001 #033 is applied, requiring **key_t** to be an arithmetic type.

The Trace option types are marked obsolescent.

The **clock_t** and **id_t** types are moved from the XSI option to the Base.

The **pthread_barrier_t** and **pthread_barrierattr_t** types are moved from the Barriers option to the Base.

The **pthread_spinlock_t** type is moved from the Spin Locks option to the Base.

Functionality relating to the Timers and Threads options is moved to the Base.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0069 [210], XBD/TC1-2008/0070 [28], XBD/TC1-2008/0071 [376], XBD/TC1-2008/0072 [210], and XBD/TC1-2008/0073 [327] are applied.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0079 [856] and XBD/TC2-2008/0080 [659] are applied.

Issue 8

Austin Group Defect 697 is applied, adding **reclen_t**.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1462 is applied, changing **time_t** to have a width of at least 64 bits.

NAME

sys/uio.h — definitions for vector I/O operations

SYNOPSIS

XSI

```
#include <sys/uio.h>
```

DESCRIPTION

The **<sys/uio.h>** header shall define the **iovec** structure, which shall include at least the following members:

`void *iov_base`	Base address of a memory region for input or output.
`size_t iov_len`	The size of the memory pointed to by *iov_base.*

The **<sys/uio.h>** header uses the **iovec** structure for scatter/gather I/O.

The **<sys/uio.h>** header shall define the **ssize_t** and **size_t** types as described in **<sys/types.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
ssize_t readv(int, const struct iovec *, int);
ssize_t writev(int, const struct iovec *, int);
```

APPLICATION USAGE

The implementation can put a limit on the number of scatter/gather elements which can be processed in one call. The symbol {IOV_MAX} defined in **<limits.h>** should always be used to learn about the limits instead of assuming a fixed value.

RATIONALE

Traditionally, the maximum number of scatter/gather elements the system can process in one call were described by the symbolic value {UIO_MAXIOV}. In IEEE Std 1003.1-2001 this value is replaced by the constant {IOV_MAX} which can be found in **<limits.h>**.

FUTURE DIRECTIONS

None.

SEE ALSO

<limits.h>, **<sys/types.h>**

XSH *read*( ), *readv*( ), *write*( ), *writev*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 6

Text referring to scatter/gather I/O is added to the DESCRIPTION.

NAME

sys/un.h — definitions for UNIX domain sockets

SYNOPSIS

```
#include <sys/un.h>
```

DESCRIPTION

The **<sys/un.h>** header shall define the **sockaddr_un** structure, which shall include at least the following members:

`sa_family_t`	`sun_family`	Address family.
`char`	`sun_path[`*size*`]`	Socket pathname storage.

The *sun_path* member shall be the last member of the **sockaddr_un** structure, where *size* shall be an implementation-provided constant size of at least 92 bytes. This *size* value need not be accessible as a constant available for use in the application namespace.

The **sockaddr_un** structure is used to store addresses for UNIX domain sockets. Pointers to this type shall be cast by applications to **struct sockaddr** * for use with socket functions.

The **<sys/un.h>** header shall define the **sa_family_t** type as described in **<sys/socket.h>**.

APPLICATION USAGE

The size of *sun_path* is required to be constant, but intentionally does not have a specified name for that constant. Historically, different implementations used different sizes. For example, 4.3 BSD used a size of 108, and 4.4 BSD used a size of 104. Since most implementations originate from BSD versions, the size is typically in the range 92 to 108. An application can deduce the size by using `sizeof(((struct sockaddr_un *)0)->sun_path)`.

Applications should not assume a particular length for *sun_path* or assume that it can hold {_POSIX_PATH_MAX} bytes (256).

Although applications are required to initialize all members (including any non-standard ones) of a **sockaddr_in6** structure (see **<netinet/in.h>**, on page 318), the same is not required for the **sockaddr_un** structure, since historically many applications only initialized the standard members. Despite this, applications are encouraged to initialize **sockaddr_un** structures in a manner similar to the required initialization of **sockaddr_in6** structures.

RATIONALE

Some implementations expose a macro SUN_LEN for the size of a pathname stored in *sun_path*. However, this was not widely adopted, and differences on how a terminating null byte is interpreted between implementations did not make it worth standardizing.

FUTURE DIRECTIONS

None.

SEE ALSO

<netinet/in.h>, **<sys/socket.h>**

XSH *bind*( ), *socket*( ), *socketpair*( )

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7

The value for {_POSIX_PATH_MAX} is updated to 256.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0074 [355] is applied.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0081 [934] is applied.

15005 **Issue 8**

15006 Austin Group Defect 561 is applied, changing the requirements for the *sun_path* member of the
15007 **sockaddr_un** structure.

15008 **NAME**

15009 sys/utsname.h — system name structure

15010 **SYNOPSIS**

15011 `#include <sys/utsname.h>`

15012 **DESCRIPTION**

15013 The **<sys/utsname.h>** header shall define the structure **utsname** which shall include at least the
15014 following members:

15015	`char sysname[]`	Name of this implementation of the operating system.
15016	`char nodename[]`	Name of this node within the communications
15017		network to which this node is attached, if any.
15018	`char release[]`	Current release level of this implementation.
15019	`char version[]`	Current version level of this release.
15020	`char machine[]`	Name of the hardware type on which the system is running.

15021 The character arrays are of unspecified size, but the data stored in them shall be terminated by a
15022 null byte.

15023 The following shall be declared as a function and may also be defined as a macro:

15024 `int uname(struct utsname *);`

15025 **APPLICATION USAGE**

15026 None.

15027 **RATIONALE**

15028 None.

15029 **FUTURE DIRECTIONS**

15030 None.

15031 **SEE ALSO**

15032 XSH *uname*( )

15033 **CHANGE HISTORY**

15034 First released in Issue 1. Derived from Issue 1 of the SVID.

15035 **Issue 6**

15036 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/27 is applied, changing the description of
15037 *nodename* within the **utsname** structure from ``an implementation-defined communications
15038 network'' to ``the communications network to which this node is attached, if any''.

15039 **NAME**

15040 sys/wait.h — declarations for waiting

15041 **SYNOPSIS**

15042 `#include <sys/wait.h>`

15043 **DESCRIPTION**

15044 The **<sys/wait.h>** header shall define the following symbolic constants for use with *waitpid*( ):

15045 XSI

WCONTINUED	Report status of continued child process.
WNOHANG	Do not hang if no status is available; return immediately.
WUNTRACED	Report status of stopped child process.

15046

15047

15048 The **<sys/wait.h>** header shall define the following macros for analysis of process status values:

WCOREDUMP	True if WIFSIGNALED is true and creation of a core image was attempted.

15049

15050

15051 **Note:** The use of the word ``attempted'' here means that the process terminated abnormally with
15052 additional actions (see SIG_DFL in XSH Section 2.4.3, on page 516). A core image might or
15053 might not have been produced. Some implementations do not set this bit if a core image was not
15054 produced, but this is not a requirement.

WEXITSTATUS	Return exit status.

15055

15056 XSI

WIFCONTINUED	True if child has been continued.

WIFEXITED	True if child exited normally.
WIFSIGNALED	True if child exited due to uncaught signal.
WIFSTOPPED	True if child stopped due to uncaught signal.
WSTOPSIG	Return signal number that caused process to stop.
WTERMSIG	Return signal number that caused process to terminate.

15057

15058

15059

15060

15061

15062 The **<sys/wait.h>** header shall define the following symbolic constants as possible values for the
15063 *options* argument to *waitid*( ):

WEXITED	Wait for processes that have terminated.
WNOWAIT	Keep the process whose status is returned in *infop* in a waitable state.
WSTOPPED	Status is returned for any child that has stopped upon receipt of a signal.

15064

15065

15066

15067 XSI The WCONTINUED and WNOHANG constants, described above for *waitpid*( ), can also be
15068 used with *waitid*( ).

15069 The type **idtype_t** shall be defined as an enumeration type whose possible values shall include
15070 at least the following:

15071 P_ALL
15072 P_PGID
15073 P_PID

15074 The **<sys/wait.h>** header shall define the **id_t** and **pid_t** types as described in **<sys/types.h>**.

15075 The **<sys/wait.h>** header shall define the **siginfo_t** type and the **sigval** union as described in
15076 **<signal.h>**.

15077 Inclusion of the **<sys/wait.h>** header may also make visible all symbols from **<signal.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
pid_t  wait(int *);
int    waitid(idtype_t, id_t, siginfo_t *, int);
pid_t  waitpid(pid_t, int *, int);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<signal.h>, **<sys/resource.h>**, **<sys/types.h>**

XSH *wait*( ), *waitid*( )

CHANGE HISTORY

First released in Issue 3.

Included for alignment with the POSIX.1-1988 standard.

Issue 6

The *wait3*( ) function is removed.

Issue 7

The *waitid*( ) function and symbolic constants for its *options* argument are moved to the Base.

The description of the WNOHANG constant is clarified.

The requirement for **<sys/wait.h>** to define the **rusage** structure as described in **<sys/resource.h>** is removed, and **<sys/wait.h>** is no longer allowed to make visible all symbols from **<sys/resource.h>**.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0082 [579] and XBD/TC2-2008/0083 [564] are applied.

Issue 8

Austin Group Defect 1141 is applied, adding WCOREDUMP and changing the description of WIFSTOPPED.

Austin Group Defect 1332 is applied, changing the description of WEXITED.

15109 **NAME**

15110 syslog.h — definitions for system error logging

15111 **SYNOPSIS**

15112 XSI
```
#include <syslog.h>
```

15113 **DESCRIPTION**

15114 The **<syslog.h>** header shall define the following symbolic constants, zero or more of which
15115 may be OR'ed together to form the *logopt* option of *openlog*( ):

15116 LOG_PID Log the process ID with each message.

15117 LOG_CONS Log to the system console on error.

15118 LOG_NDELAY Connect to syslog daemon immediately.

15119 LOG_ODELAY Delay open until *syslog*( ) is called.

15120 LOG_NOWAIT Do not wait for child processes.

15121 The **<syslog.h>** header shall define the following symbolic constants for use as the *facility*
15122 argument to *openlog*( ):

15123 LOG_KERN Reserved for message generated by the system.

15124 LOG_USER Message generated by a process.

15125 LOG_MAIL Reserved for message generated by mail system.

15126 LOG_NEWS Reserved for message generated by news system.

15127 LOG_UUCP Reserved for message generated by UUCP system.

15128 LOG_DAEMON Reserved for message generated by system daemon.

15129 LOG_AUTH Reserved for message generated by authorization daemon.

15130 LOG_CRON Reserved for message generated by clock daemon.

15131 LOG_LPR Reserved for message generated by printer system.

15132 LOG_LOCAL0 Reserved for local use.

15133 LOG_LOCAL1 Reserved for local use.

15134 LOG_LOCAL2 Reserved for local use.

15135 LOG_LOCAL3 Reserved for local use.

15136 LOG_LOCAL4 Reserved for local use.

15137 LOG_LOCAL5 Reserved for local use.

15138 LOG_LOCAL6 Reserved for local use.

15139 LOG_LOCAL7 Reserved for local use.

15140 The **<syslog.h>** header shall define the following macros for constructing the *maskpri* argument
15141 to *setlogmask*( ). The following macros expand to an expression of type **int** when the argument
15142 *pri* is an expression of type **int**:

15143 LOG_MASK(*pri*) A mask for priority *pri*.

LOG_UPTO(*pri*) A mask for all priorities from LOG_EMERG through *pri* inclusive in the order listed in the list of priority symbolic constants below. Any additional implementation-defined priorities not included in the list below shall not be included in the mask.

The **<syslog.h>** header shall define the following symbolic constants for use as the severity level portion of the *priority* argument of *syslog*( ) and the *pri* argument of the LOG_MASK(*pri*) and LOG_UPTO(*pri*) macros:

LOG_EMERG A panic condition was reported to all processes.

LOG_ALERT A condition that should be corrected immediately.

LOG_CRIT A critical condition.

LOG_ERR An error message.

LOG_WARNING A warning message.

LOG_NOTICE A condition requiring special handling.

LOG_INFO A general information message.

LOG_DEBUG A message useful for debugging programs.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
void  closelog(void);
void  openlog(const char *, int, int);
int   setlogmask(int);
void  syslog(int, const char *, ...);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH *closelog*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/Open UNIX to BASE.

Issue 7

This reference page is clarified with respect to macros and symbolic constants.

Issue 8

Austin Group Defect 1033 is applied, adding the LOG_UPTO macro.

15181 **NAME**

15182 tar.h — extended tar definitions

15183 **SYNOPSIS**

15184 `#include <tar.h>`

15185 **DESCRIPTION**

15186 The **<tar.h>** header shall define the following symbolic constants with the indicated values.

15187 General definitions:

15188

Name	Value	Description
15189 TMAGIC	`"ustar"`	Used in the magic field in the **ustar** header block, including the trailing null byte.
15191 TMAGLEN	6	Length in octets of the magic field.
15192 TVERSION	`"00"`	Used in the version field in the **ustar** header block, excluding the trailing null byte.
15194 TVERSLEN	2	Length in octets of the version field.

15195 *Typeflag* field definitions:

15196

Name	Value	Description
15197 REGTYPE	`'0'`	Regular file.
15198 AREGTYPE	`'\0'`	Regular file.
15199 LNKTYPE	`'1'`	Hard Link.
15200 SYMTYPE	`'2'`	Symbolic link.
15201 CHRTYPE	`'3'`	Character special.
15202 BLKTYPE	`'4'`	Block special.
15203 DIRTYPE	`'5'`	Directory.
15204 FIFOTYPE	`'6'`	FIFO special.
15205 CONTTYPE	`'7'`	Reserved.

15206 *Mode* field bit definitions (octal):

15207

Name	Value	Description
15208 TSUID	04000	Set UID on execution.
15209 TSGID	02000	Set GID on execution.
15210 XSI TSVTX	01000	On directories, restricted deletion flag.
15211 TUREAD	00400	Read by owner.
15212 TUWRITE	00200	Write by owner special.
15213 TUEXEC	00100	Execute/search by owner.
15214 TGREAD	00040	Read by group.
15215 TGWRITE	00020	Write by group.
15216 TGEXEC	00010	Execute/search by group.
15217 TOREAD	00004	Read by other.
15218 TOWRITE	00002	Write by other.
15219 TOEXEC	00001	Execute/search by other.

15220 **APPLICATION USAGE**

15221 None.

15222 **RATIONALE**

15223 None.

15224 **FUTURE DIRECTIONS**

15225 None.

15226 **SEE ALSO**

15227 XCU *pax*

15228 **CHANGE HISTORY**

15229 First released in Issue 3. Derived from the POSIX.1-1988 standard.

15230 **Issue 6**

15231 The SEE ALSO section is updated to refer to *pax*.

15232 **Issue 7**

15233 This reference page is clarified with respect to macros and symbolic constants.

15234 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0084 [707] is applied.

15235 **Issue 8**

15236 Austin Group Defect 1380 is applied, changing the description of LNKTYPE.

15237 **NAME**

15238 termios.h — define values for termios

15239 **SYNOPSIS**

15240 `#include <termios.h>`

15241 **DESCRIPTION**

15242 The **<termios.h>** header shall contain the definitions used by the terminal I/O interfaces (see
15243 Chapter 11 (on page 199) for the structures and names defined).

15244 **The termios Structure**

15245 The **<termios.h>** header shall define the following data types through **typedef**:

15246 **cc_t** Used for terminal special characters.

15247 **speed_t** Used for terminal baud rates.

15248 **tcflag_t** Used for terminal modes.

15249 The above types shall be all unsigned integer types.

15250 The implementation shall support one or more programming environments in which the widths
15251 of **cc_t**, **speed_t**, and **tcflag_t** are no greater than the width of type **long**. The names of these
15252 programming environments can be obtained using the *confstr*( ) function or the *getconf* utility.

15253 The **<termios.h>** header shall define the **termios** structure, which shall include at least the
15254 following members:

15255 `tcflag_t  c_iflag` Input modes.
15256 `tcflag_t  c_oflag` Output modes.
15257 `tcflag_t  c_cflag` Control modes.
15258 `tcflag_t  c_lflag` Local modes.
15259 `cc_t  c_cc[NCCS]` Control characters.

15260 The **<termios.h>** header shall define the following symbolic constant:

15261 NCCS Size of the array *c_cc* for control characters.

15262 The **<termios.h>** header shall define the following symbolic constants for use as subscripts for
15263 the array *c_cc*:

15264 15265

Subscript Usage		
Canonical Mode	**Non-Canonical Mode**	**Description**
15266 VEOF		EOF character.
15267 VEOL		EOL character.
15268 VERASE		ERASE character.
15269 VINTR	VINTR	INTR character.
15270 VKILL		KILL character.
15271	VMIN	MIN value.
15272 VQUIT	VQUIT	QUIT character.
15273 VSTART	VSTART	START character.
15274 VSTOP	VSTOP	STOP character.
15275 VSUSP	VSUSP	SUSP character.
15276	VTIME	TIME value.

15277 The subscript values shall be suitable for use in **#if** preprocessing directives and shall be distinct,
15278 except that the VMIN and VTIME subscripts may have the same values as the VEOF and VEOL
15279 subscripts, respectively.

15280 **Input Modes**

15281 The **<termios.h>** header shall define the following symbolic constants for use as flags in the
15282 *c_iflag* field. The *c_iflag* field describes the basic terminal input control.

15283 BRKINT Signal interrupt on break.

15284 ICRNL Map CR to NL on input.

15285 IGNBRK Ignore break condition.

15286 IGNCR Ignore CR.

15287 IGNPAR Ignore characters with parity errors.

15288 INLCR Map NL to CR on input.

15289 INPCK Enable input parity check.

15290 ISTRIP Strip character.

15291 IXANY Enable any character to restart output.

15292 IXOFF Enable start/stop input control.

15293 IXON Enable start/stop output control.

15294 PARMRK Mark parity errors.

15295 **Output Modes**

15296 The **<termios.h>** header shall define the following symbolic constants for use as flags in the
15297 *c_oflag* field. The *c_oflag* field specifies the system treatment of output.

15298 OPOST Post-process output.

15299 XSI ONLCR Map NL to CR-NL on output.

15300 XSI OCRNL Map CR to NL on output.

15301 XSI ONOCR No CR output at column 0.

15302 XSI ONLRET NL performs CR function.

15303 XSI OFDEL Fill is DEL.

15304 XSI OFILL Use fill characters for delay.

15305 XSI NLDLY Select newline delays:

15306 NL0 Newline type 0.

15307 NL1 Newline type 1.

15308 XSI CRDLY Select carriage-return delays:

15309 CR0 Carriage-return delay type 0.

15310 CR1 Carriage-return delay type 1.

15311 CR2 Carriage-return delay type 2.

15312 CR3 Carriage-return delay type 3.

15313 XSI

TABDLY	Select horizontal-tab delays:
	TAB0 Horizontal-tab delay type 0.
	TAB1 Horizontal-tab delay type 1.
	TAB2 Horizontal-tab delay type 2.
	TAB3 Expand tabs to spaces.

15318 XSI

BSDLY	Select backspace delays:
	BS0 Backspace-delay type 0.
	BS1 Backspace-delay type 1.

15321 XSI

VTDLY	Select vertical-tab delays:
	VT0 Vertical-tab delay type 0.
	VT1 Vertical-tab delay type 1.

15324 XSI

FFDLY	Select form-feed delays:
	FF0 Form-feed delay type 0.
	FF1 Form-feed delay type 1.

15327 **Baud Rate Selection**

15328 The **<termios.h>** header shall define the following symbolic constants for use as values of
15329 objects of type **speed_t**.

15330 The input and output baud rates are stored in the **termios** structure. These are the valid values
15331 for objects of type **speed_t**. Not all baud rates need be supported by the underlying hardware.

15332 B0	Hang up
15333 B50	50 baud
15334 B75	75 baud
15335 B110	110 baud
15336 B134	134.5 baud
15337 B150	150 baud
15338 B200	200 baud
15339 B300	300 baud
15340 B600	600 baud
15341 B1200	1 200 baud
15342 B1800	1 800 baud
15343 B2400	2 400 baud

B4800	4 800 baud
B9600	9 600 baud
B19200	19 200 baud
B38400	38 400 baud

Control Modes

The **<termios.h>** header shall define the following symbolic constants for use as flags in the *c_cflag* field. The *c_cflag* field describes the hardware control of the terminal; not all values specified are required to be supported by the underlying hardware.

CSIZE	Character size:	
	CS5	5 bits
	CS6	6 bits
	CS7	7 bits
	CS8	8 bits
CSTOPB	Send two stop bits, else one.	
CREAD	Enable receiver.	
PARENB	Parity enable.	
PARODD	Odd parity, else even.	
HUPCL	Hang up on last close.	
CLOCAL	Ignore modem status lines.	

The implementation shall support the functionality associated with the symbols CS7, CS8, CSTOPB, PARODD, and PARENB.

Local Modes

The **<termios.h>** header shall define the following symbolic constants for use as flags in the *c_lflag* field. The *c_lflag* field of the argument structure is used to control various terminal functions.

ECHO	Enable echo.
ECHOE	Echo erase character as error-correcting backspace.
ECHOK	Echo KILL.
ECHONL	Echo NL.
ICANON	Canonical input (erase and kill processing).
IEXTEN	Enable extended input character processing.
ISIG	Enable signals.
NOFLSH	Disable flush after interrupt or quit.
TOSTOP	Send SIGTTOU for background output.

15378 **The winsize Structure**

15379 The **<termios.h>** header shall define the **winsize** structure, which shall include at least the
15380 following members:

15381 `unsigned short ws_row` Rows, in characters.
15382 `unsigned short ws_col` Columns, in characters.

15383 **Attribute Selection**

15384 The **<termios.h>** header shall define the following symbolic constants for use with *tcsetattr*( ):

15385 TCSANOW Change attributes immediately.

15386 TCSADRAIN Change attributes when output has drained.

15387 TCSAFLUSH Change attributes when output has drained; also flush pending input.

15388 **Line Control**

15389 The **<termios.h>** header shall define the following symbolic constants for use with *tcflush*( ):

15390 TCIFLUSH Flush pending input.

15391 TCIOFLUSH Flush both pending input and untransmitted output.

15392 TCOFLUSH Flush untransmitted output.

15393 The **<termios.h>** header shall define the following symbolic constants for use with *tcflow*( ):

15394 TCIOFF Transmit a STOP character, intended to suspend input data.

15395 TCION Transmit a START character, intended to restart input data.

15396 TCOOFF Suspend output.

15397 TCOON Restart output.

15398 The **<termios.h>** header shall define the **pid_t** type as described in **<sys/types.h>**.

15399 The following shall be declared as functions and may also be defined as macros. Function
15400 prototypes shall be provided.

```
15401 speed_t cfgetispeed(const struct termios *);
15402 speed_t cfgetospeed(const struct termios *);
15403 int     cfsetispeed(struct termios *, speed_t);
15404 int     cfsetospeed(struct termios *, speed_t);
15405 int     tcdrain(int);
15406 int     tcflow(int, int);
15407 int     tcflush(int, int);
15408 int     tcgetattr(int, struct termios *);
15409 pid_t   tcgetsid(int);
15410 int     tcgetwinsize(int, struct winsize *);
15411 int     tcsendbreak(int, int);
15412 int     tcsetattr(int, int, const struct termios *);
15413 int     tcsetwinsize(int, const struct winsize *);
```


APPLICATION USAGE

The following names are reserved for XSI-conformant systems to use as an extension to the above; therefore strictly conforming applications shall not use them:

CBAUD	EXTB	VDSUSP
DEFECHO	FLUSHO	VLNEXT
ECHOCTL	LOBLK	VREPRINT
ECHOKE	PENDIN	VSTATUS
ECHOPRT	SWTCH	VWERASE
EXTA	VDISCARD	

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>

XSH *cfgetispeed*( ), *cfgetospeed*( ), *cfsetispeed*( ), *cfsetospeed*( ), *confstr*( ), *tcdrain*( ), *tcflow*( ), *tcflush*( ), *tcgetattr*( ), *tcgetsid*( ), *tcgetwinsize*( ), *tcsendbreak*( ), *tcsetattr*( ), *tcsetwinsize*( )

XCU Chapter 11 (on page 199), *getconf*

CHANGE HISTORY

First released in Issue 3.

Included for alignment with the ISO POSIX-1 standard.

Issue 6

The LEGACY symbols IUCLC, OLCUC, and XCASE are removed.

FIPS 151-2 requirements for the symbols CS7, CS8, CSTOPB, PARODD, and PARENB are reaffirmed.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/19 is applied, changing ECHOK to ECHOKE in the APPLICATION USAGE section.

Issue 7

Austin Group Interpretation 1003.1-2001 #144 is applied, moving functionality relating to the IXANY symbol from the XSI option to the Base.

SD5-XBD-ERN-35 is applied, adding the OFDEL output mode.

This reference page is clarified with respect to macros and symbolic constants, and a declaration for the **pid_t** type is added.

Issue 8

Austin Group Defects 1151 and 1484 are applied, adding the **winsize** structure and the *tcgetwinsize*( ) and *tcsetwinsize*( ) functions.

15450 **NAME**

15451 tgmath.h — type-generic macros

15452 **SYNOPSIS**

15453 `#include <tgmath.h>`

15454 **DESCRIPTION**

15455 CX The functionality described on this reference page is aligned with the ISO C standard. Any
15456 conflict between the requirements described here and the ISO C standard is unintentional. This
15457 volume of POSIX.1-2024 defers to the ISO C standard.

15458 The **<tgmath.h>** header shall include the headers **<math.h>** and **<complex.h>** and shall define
15459 several type-generic macros.

15460 Of the functions contained within the **<math.h>** and **<complex.h>** headers without an *f* (**float**)
15461 or *l* (**long double**) suffix, several have one or more parameters whose corresponding real type is
15462 XSI **double**. For each such function, except *modf*( ), *j0*( ), *j1*( ), *jn*( ), *y0*( ), *y1*( ), and *yn*( ), there shall
15463 be a corresponding type-generic macro. The parameters whose corresponding real type is
15464 **double** in the function synopsis are generic parameters. Use of the macro invokes a function
15465 whose corresponding real type and type domain are determined by the arguments for the
15466 generic parameters.

15467 Use of the macro invokes a function whose generic parameters have the corresponding real type
15468 determined as follows:

15469 • First, if any argument for generic parameters has type **long double**, the type determined is
15470 **long double**.

15471 • Otherwise, if any argument for generic parameters has type **double** or is of integer type,
15472 the type determined is **double**.

15473 • Otherwise, the type determined is **float**.

15474 For each unsuffixed function in the **<math.h>** header for which there is a function in the
15475 **<complex.h>** header with the same name except for a *c* prefix, the corresponding type-generic
15476 macro (for both functions) has the same name as the function in the **<math.h>** header. The
15477 corresponding type-generic macro for *fabs*( ) and *cabs*( ) is *fabs*( ).

15478

<math.h> Function	<complex.h> Function	Type-Generic Macro
15479 *acos*( )	*cacos*( )	*acos*( )
15480 *asin*( )	*casin*( )	*asin*( )
15481 *atan*( )	*catan*( )	*atan*( )
15482 *acosh*( )	*cacosh*( )	*acosh*( )
15483 *asinh*( )	*casinh*( )	*asinh*( )
15484 *atanh*( )	*catanh*( )	*atanh*( )
15485 *cos*( )	*ccos*( )	*cos*( )
15486 *sin*( )	*csin*( )	*sin*( )
15487 *tan*( )	*ctan*( )	*tan*( )
15488 *cosh*( )	*ccosh*( )	*cosh*( )
15489 *sinh*( )	*csinh*( )	*sinh*( )
15490 *tanh*( )	*ctanh*( )	*tanh*( )
15491 *exp*( )	*cexp*( )	*exp*( )
15492 *log*( )	*clog*( )	*log*( )
15493 *pow*( )	*cpow*( )	*pow*( )
15494 *sqrt*( )	*csqrt*( )	*sqrt*( )
15495 *fabs*( )	*cabs*( )	*fabs*( )

If at least one argument for a generic parameter is complex, then use of the macro invokes a complex function; otherwise, use of the macro invokes a real function.

For each unsuffixed function in the **<math.h>** header without a *c*-prefixed counterpart in the **<complex.h>** header, except for *modf*( ), *j0*( ), *j1*( ), *jn*( ), *y0*( ), *y1*( ), and *yn*( ), the corresponding type-generic macro has the same name as the function. These type-generic macros are:

atan2( )	*fma*( )	*llround*( )	*remainder*( )
cbrt( )	*fmax*( )	*log10*( )	*remquo*( )
ceil( )	*fmin*( )	*log1p*( )	*rint*( )
copysign( )	*fmod*( )	*log2*( )	*round*( )
erf( )	*frexp*( )	*logb*( )	*scalbln*( )
erfc( )	*hypot*( )	*lrint*( )	*scalbn*( )
exp2( )	*ilogb*( )	*lround*( )	*tgamma*( )
expm1( )	*ldexp*( )	*nearbyint*( )	*trunc*( )
fdim( )	*lgamma*( )	*nextafter*( )	
floor( )	*llrint*( )	*nexttoward*( )	

If all arguments for generic parameters are real, then use of the macro invokes a real function; otherwise, use of the macro results in undefined behavior.

For each unsuffixed function in the **<complex.h>** header that is not a *c*-prefixed counterpart to a function in the **<math.h>** header, the corresponding type-generic macro has the same name as the function. These type-generic macros are:

carg( )
cimag( )
conj( )
cproj( )
creal( )

Use of the macro with any real or complex argument invokes a complex function.

MXC Type-generic macros that accept complex arguments shall also accept imaginary arguments. If an argument is imaginary, the macro shall expand to an expression whose type is real, imaginary, or complex, as appropriate for the particular function: if the argument is imaginary, then the types of *cos*( ), *cosh*( ), *fabs*( ), *carg*( ), *cimag*( ), and *creal*( ) shall be real; the types of *sin*( ), *tan*( ), *sinh*( ), *tanh*( ), *asin*( ), *atan*( ), *asinh*( ), and *atanh*( ) shall be imaginary; and the types of the others shall be complex.

Given an imaginary argument, each of the type-generic macros *cos*( ), *sin*( ), *tan*( ), *cosh*( ), *sinh*( ), *tanh*( ), *asin*( ), *atan*( ), *asinh*( ), and *atanh*( ) is specified by a formula in terms of real functions:

$$\begin{aligned}
\cos(iy) &= \cosh(y) \\
\sin(iy) &= i\sinh(y) \\
\tan(iy) &= i\tanh(y) \\
\cosh(iy) &= \cos(y) \\
\sinh(iy) &= i\sin(y) \\
\tanh(iy) &= i\tan(y) \\
\operatorname{asin}(iy) &= i\operatorname{asinh}(y) \\
\operatorname{atan}(iy) &= i\operatorname{atanh}(y) \\
\operatorname{asinh}(iy) &= i\operatorname{asin}(y) \\
\operatorname{atanh}(iy) &= i\operatorname{atan}(y)
\end{aligned}$$

APPLICATION USAGE

With the declarations:

```
#include <tgmath.h>
int n;
float f;
double d;
long double ld;
float complex fc;
double complex dc;
long double complex ldc;
```

functions invoked by use of type-generic macros are shown in the following table:

Macro	Use Invokes
exp(*n*)	*exp*(*n*), the function
acosh(*f*)	*acoshf*(*f*)
sin(*d*)	*sin*(*d*), the function
atan(*ld*)	*atanl*(*ld*)
log(*fc*)	*clogf*(*fc*)
sqrt(*dc*)	*csqrt*(*dc*)
pow(*ldc,f*)	*cpowl*(*ldc, f*)
remainder(*n,n*)	*remainder*(*n, n*), the function
nextafter(*d,f*)	*nextafter*(*d, f*), the function
nexttoward(*f,ld*)	*nexttowardf*(*f, ld*)
copysign(*n,ld*)	*copysignl*(*n, ld*)
ceil(*fc*)	Undefined behavior
rint(*dc*)	Undefined behavior
fmax(*ldc,ld*)	Undefined behavior
carg(*n*)	*carg*(*n*), the function
cproj(*f*)	*cprojf*(*f*)
creal(*d*)	*creal*(*d*), the function
cimag(*ld*)	*cimagl*(*ld*)
cabs(*fc*)	*cabsf*(*fc*)
carg(*dc*)	*carg*(*dc*), the function
cproj(*ldc*)	*cprojl*(*ldc*)

RATIONALE

Type-generic macros allow calling a function whose type is determined by the argument type, as is the case for C operators such as `'+'` and `'*'`. For example, with a type-generic *cos*( ) macro, the expression *cos*((**float**)*x*) will have type **float**. This feature enables writing more portably efficient code and alleviates need for awkward casting and suffixing in the process of porting or adjusting precision. Generic math functions are a widely appreciated feature of Fortran.

The only arguments that affect the type resolution are the arguments corresponding to the parameters that have type **double** in the synopsis. Hence the type of a type-generic call to *nexttoward*( ), whose second parameter is **long double** in the synopsis, is determined solely by the type of the first argument.

The term ``type-generic'' was chosen over the proposed alternatives of intrinsic and overloading. The term is more specific than intrinsic, which already is widely used with a more general meaning, and reflects a closer match to Fortran's generic functions than to C++ overloading.

The macros are placed in their own header in order not to silently break old programs that

include the **<math.h>** header; for example, with:

```
printf ("%e", sin(x))
```

modf(**double**, **double** *) is excluded because no way was seen to make it safe without complicating the type resolution.

The implementation might, as an extension, endow appropriate ones of the macros that POSIX.1-2024 specifies only for real arguments with the ability to invoke the complex functions.

POSIX.1-2024 does not prescribe any particular implementation mechanism for generic macros. It could be implemented simply with built-in macros. The generic macro for *sqrt*(), for example, could be implemented with:

```
#undef sqrt
#define sqrt(x) __BUILTIN_GENERIC_sqrt(x)
```

Generic macros are designed for a useful level of consistency with C++ overloaded math functions.

The great majority of existing C programs are expected to be unaffected when the **<tgmath.h>** header is included instead of the **<math.h>** or **<complex.h>** headers. Generic macros are similar to the ISO C standard library masking macros, though the semantic types of return values differ.

The ability to overload on integer as well as floating types would have been useful for some functions; for example, *copysign*(). Overloading with different numbers of arguments would have allowed reusing names; for example, *remainder*() for *remquo*(). However, these facilities would have complicated the specification; and their natural consistent use, such as for a floating *abs*() or a two-argument *atan*(), would have introduced further inconsistencies with the ISO/IEC 9899: 1999 standard for insufficient benefit.

The ISO C standard in no way limits the implementation's options for efficiency, including inlining library functions.

FUTURE DIRECTIONS

None.

SEE ALSO

<math.h>, **<complex.h>**

XSH *cabs*(), *fabs*(), *modf*()

CHANGE HISTORY

First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #184 is applied, clarifying the functions for which a corresponding type-generic macro exists with the same name as the function.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0075 [357,427] is applied.

Issue 8

Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899: 2018 standard.

NAME

threads.h — ISO C threads

SYNOPSIS

```
#include <threads.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

Implementations shall not define the macro __STDC_NO_THREADS__, except for profile implementations that define _POSIX_SUBPROFILE (see Section 2.1.5.1, on page 20) in **<unistd.h>**, which may define __STDC_NO_THREADS__ and, if they do so, need not provide this header nor support any of its facilities.

The **<threads.h>** header shall define the following macros:

thread_local Expands to **_Thread_local**.

ONCE_FLAG_INIT Expands to a value that can be used to initialize an object of type **once_flag**.

TSS_DTOR_ITERATIONS
Expands to an integer constant expression representing the maximum number of times that destructors will be called when a thread terminates and shall be suitable for use in **#if** preprocessing directives.

CX If {PTHREAD_DESTRUCTOR_ITERATIONS} is defined in **<limits.h>**, the value of TSS_DTOR_ITERATIONS shall be equal to {PTHREAD_DESTRUCTOR_ITERATIONS}; otherwise, the value of TSS_DTOR_ITERATIONS shall be greater than or equal to the value of {_POSIX_THREAD_DESTRUCTOR_ITERATIONS} and shall be less than or equal to the maximum positive value that can be returned by a call to *sysconf* (_SC_THREAD_DESTRUCTOR_ITERATIONS) in any process.

The **<threads.h>** header shall define the types **cnd_t**, **mtx_t**, **once_flag**, **thrd_t**, and **tss_t** as complete object types, the type **thrd_start_t** as the function pointer type **int (*)(void*)**, and the CX type **tss_dtor_t** as the function pointer type **void (*)(void*)**. The type **thrd_t** shall be defined to be the same type that **pthread_t** is defined to be in **<pthread.h>**.

The **<threads.h>** header shall define the enumeration constants `mtx_plain`, `mtx_recursive`, `mtx_timed`, `thrd_busy`, `thrd_error`, `thrd_nomem`, `thrd_success` and `thrd_timedout`.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
void            call_once(once_flag *, void (*)(void));
int             cnd_broadcast(cnd_t *);
void            cnd_destroy(cnd_t *);
int             cnd_init(cnd_t *);
int             cnd_signal(cnd_t *);
int             cnd_timedwait(cnd_t * restrict, mtx_t * restrict,
                    const struct timespec * restrict);
int             cnd_wait(cnd_t *, mtx_t *);
void            mtx_destroy(mtx_t *);
int             mtx_init(mtx_t *, int);
int             mtx_lock(mtx_t *);
int             mtx_timedlock(mtx_t * restrict,
```



```
                        const struct timespec * restrict);
int             mtx_trylock(mtx_t *);
int             mtx_unlock(mtx_t *);
int             thrd_create(thrd_t *, thrd_start_t, void *);
thrd_t          thrd_current(void);
int             thrd_detach(thrd_t);
int             thrd_equal(thrd_t, thrd_t);
_Noreturn void  thrd_exit(int);
int             thrd_join(thrd_t, int *);
int             thrd_sleep(const struct timespec *, struct timespec *);
void            thrd_yield(void);
int             tss_create(tss_t *, tss_dtor_t);
void            tss_delete(tss_t);
void           *tss_get(tss_t);
int             tss_set(tss_t, void *);
```

Inclusion of the **<threads.h>** header shall make symbols defined in the header **<time.h>** visible.

APPLICATION USAGE

The **<threads.h>** header is optional in the ISO C standard but is mandated by POSIX.1-2024. Note however that subprofiles can choose to make this header optional (see Section 2.1.5.1, on page 20), and therefore application portability to subprofile implementations would benefit from checking whether __STDC_NO_THREADS__ is defined before inclusion of **<threads.h>**.

The features provided by **<threads.h>** are not as extensive as those provided by **<pthread.h>**. It is present on POSIX.1 implementations in order to facilitate porting of ISO C programs that use it. It is recommended that applications intended for use on POSIX.1 implementations use **<pthread.h>** rather than **<threads.h>** even if none of the additional features are needed initially, to save the need to convert should the need to use them arise later in the application's lifecycle.

RATIONALE

Although the **<threads.h>** header is optional in the ISO C standard, it is mandated by POSIX.1-2024 because **<pthread.h>** is mandatory and the interfaces in **<threads.h>** can easily be implemented as a thin wrapper for interfaces in **<pthread.h>**.

The type **thrd_t** is required to be defined as the same type that **pthread_t** is defined to be in **<pthread.h>** because *thrd_current*() and *pthread_self*() need to return the same thread ID when called from the initial thread. However, these types are not fully interchangeable (that is, it is not always possible to pass a thread ID obtained as a **thrd_t** to a function that takes a **pthread_t**, and vice versa) because threads created using *thrd_create*() have a different exit status than *pthreads* threads, which is reflected in differences between the prototypes for *thrd_create*() and *pthread_create*(), *thrd_exit*() and *pthread_exit*(), and *thrd_join*() and *pthread_join*(); also, *thrd_join*() has no way to indicate that a thread was cancelled.

The standard developers considered making it implementation-defined whether the types **cnd_t, mtx_t** and **tss_t** are interchangeable with the corresponding types **pthread_cond_t, pthread_mutex_t** and **pthread_key_t** defined in **<pthread.h>** (that is, whether any function that can be called with a valid **cnd_t** can also be called with a valid **pthread_cond_t,** and vice versa, and likewise for the other types). However, this would have meant extending *mtx_lock*() to provide a way for it to indicate that the owner of a mutex has terminated (equivalent to [EOWNERDEAD]). It was felt that such an extension would be invention. Although there was no similar concern for **cnd_t** and **tss_t,** they were treated the same way as **mtx_t** for consistency. See also the RATIONALE for *mtx_lock*() concerning the inability of **mtx_t** to contain information about whether or not a mutex supports timeout if it is the same type as **pthread_mutex_t.**

FUTURE DIRECTIONS

None.

SEE ALSO

<limits.h>, **<pthread.h>**, **<time.h>**

XSH Section 2.9, *call_once*( ), *cnd_broadcast*( ), *cnd_destroy*( ), *cnd_timedwait*( ), *mtx_destroy*( ), *mtx_lock*( ), *sysconf*( ), *thrd_create*( ), *thrd_current*( ), *thrd_detach*( ), *thrd_equal*( ), *thrd_exit*( ), *thrd_join*( ), *thrd_sleep*( ), *thrd_yield*( ), *tss_create*( ), *tss_delete*( ), *tss_get*( )

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

time.h — time types

SYNOPSIS

```
#include <time.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to enable the visibility of these symbols in this header.

The **<time.h>** header shall define the **clock_t**, **size_t**, **time_t**, types as described in **<sys/types.h>**.

CX The **<time.h>** header shall define the **clockid_t** and **timer_t** types as described in **<sys/types.h>**.

The **<time.h>** header shall define the **locale_t** type as described in **<locale.h>**.

CPT The **<time.h>** header shall define the **pid_t** type as described in **<sys/types.h>**.

CX The tag **sigevent** shall be declared as naming an incomplete structure type, the contents of which are described in the **<signal.h>** header.

The **<time.h>** header shall declare the **tm** structure, which shall include at least the following members:

`int`	`tm_sec`	Seconds [0,60].
`int`	`tm_min`	Minutes [0,59].
`int`	`tm_hour`	Hour [0,23].
`int`	`tm_mday`	Day of month [1,31].
`int`	`tm_mon`	Month of year [0,11].
`int`	`tm_year`	Years since 1900.
`int`	`tm_wday`	Day of week [0,6] (Sunday =0).
`int`	`tm_yday`	Day of year [0,365].
`int`	`tm_isdst`	Daylight Saving flag.
`long`	`tm_gmtoff`	Seconds east of UTC.
`const char *`	`tm_zone`	Timezone abbreviation.

When *tm_isdst* is set by an interface defined in this standard, its value shall be positive if CX Daylight Saving Time (DST) is in effect and 0 if DST is not in effect. It shall not be set to a negative value by any interface defined in this standard. When *tm_isdst* is passed to the *mktime*( ) function, it specifies how *mktime*( ) is to handle DST when calculating the time since the Epoch value; see *mktime*( ).

CX If the value of *tm_zone* is accessed after the value of *TZ* is subsequently modified, and the *tm_zone* value was not set by a call to *gmtime*( ) or *gmtime_r*( ), the behavior is undefined.

The **<time.h>** header shall declare the **timespec** structure, which shall include at least the following members:

`time_t`	`tv_sec`	Whole seconds.
`long`	`tv_nsec`	Nanoseconds [0, 999 999 999].

CX The **<time.h>** header shall also declare the **itimerspec** structure, which shall include at least the following members:

`struct timespec`	`it_interval`	Timer period.
`struct timespec`	`it_value`	Timer expiration.

The **<time.h>** header shall define the following macros:

NULL — As described in **<stddef.h>**.

CLOCKS_PER_SEC — A number used to convert the value returned by the *clock*( ) function into seconds. The value shall be an expression with type **clock_t**. The value of CLOCKS_PER_SEC shall be 1 million on XSI-conformant systems. However, it may be variable on other systems, and it should not be assumed that CLOCKS_PER_SEC is a compile-time constant.

TIME_UTC — An integer constant greater than 0 that designates the UTC time base in calls to *timespec_get*( ). The value shall be suitable for use in **#if** preprocessing directives.

The **<time.h>** header shall define the following symbolic constants. The values shall have a type that is assignment-compatible with **clockid_t**.

CLOCK_MONOTONIC
The identifier for the system-wide monotonic clock, which is defined as a clock measuring real time, whose value cannot be set via *clock_settime*( ) and which cannot have negative clock jumps. The maximum possible clock jump shall be implementation-defined.

CLOCK_PROCESS_CPUTIME_ID
The identifier of the CPU-time clock associated with the process making a *clock**( ) or *timer**( ) function call.

CLOCK_REALTIME — The identifier of the system-wide clock measuring real time.

CLOCK_THREAD_CPUTIME_ID
The identifier of the CPU-time clock associated with the thread making a *clock**( ) or *timer**( ) function call.

The **<time.h>** header shall define the following symbolic constant:

TIMER_ABSTIME — Flag indicating time is absolute. For functions taking timer objects, this refers to the clock associated with the timer.

The **<time.h>** header shall provide a declaration or definition for *getdate_err*. The *getdate_err* symbol shall expand to an expression of type **int**. It is unspecified whether *getdate_err* is a macro or an identifier declared with external linkage, and whether or not it is a modifiable lvalue. If a macro definition is suppressed in order to access an actual object, or a program defines an identifier with the name *getdate_err*, the behavior is undefined.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
char      *asctime(const struct tm *);
clock_t    clock(void);
int        clock_getcpuclockid(pid_t, clockid_t *);
int        clock_getres(clockid_t, struct timespec *);
int        clock_gettime(clockid_t, struct timespec *);
int        clock_nanosleep(clockid_t, int, const struct timespec *,
               struct timespec *);
int        clock_settime(clockid_t, const struct timespec *);
char      *ctime(const time_t *);
double     difftime(time_t, time_t);
```



```
15814 XSI   struct tm *getdate(const char *);
15815       struct tm *gmtime(const time_t *);
15816 CX    struct tm *gmtime_r(const time_t *restrict, struct tm *restrict);
15817       struct tm *localtime(const time_t *);
15818 CX    struct tm *localtime_r(const time_t *restrict, struct tm *restrict);
15819       time_t     mktime(struct tm *);
15820 CX    int        nanosleep(const struct timespec *, struct timespec *);
15821       size_t     strftime(char *restrict, size_t, const char *restrict,
15822                  const struct tm *restrict);
15823 CX    size_t     strftime_l(char *restrict, size_t, const char *restrict,
15824                      const struct tm *restrict, locale_t);
15825 XSI   char      *strptime(const char *restrict, const char *restrict,
15826                      struct tm *restrict);
15827       time_t     time(time_t *);
15828 CX    int        timer_create(clockid_t, struct sigevent *restrict,
15829                      timer_t *restrict);
15830       int        timer_delete(timer_t);
15831       int        timer_getoverrun(timer_t);
15832       int        timer_gettime(timer_t, struct itimerspec *);
15833       int        timer_settime(timer_t, int, const struct itimerspec *restrict,
15834                      struct itimerspec *restrict);
15835       int        timespec_get(struct timespec *, int);
15836 CX    void       tzset(void);
```

15837 The **<time.h>** header shall declare the following as variables:

```
15838 XSI   extern int    daylight;
15839       extern long   timezone;
15840 CX    extern char  *tzname[];
```

15841 CX Inclusion of the **<time.h>** header may make visible all symbols from the **<signal.h>** header.

15842 **APPLICATION USAGE**

15843 The range [0,60] for *tm_sec* allows for the occasional leap second.

15844 *tm_year* is a signed value; therefore, years before 1900 may be represented.

15845 To obtain the number of clock ticks per second returned by the *times*() function, applications
15846 should call *sysconf*(_SC_CLK_TCK).

15847 **RATIONALE**

15848 The range [0,60] seconds allows for positive or negative leap seconds. The formal definition of
15849 UTC does not permit double leap seconds, so all mention of double leap seconds has been
15850 removed, and the range shortened from the former [0,61] seconds seen in earlier versions of this
15851 standard.

15852 **FUTURE DIRECTIONS**

15853 None.

15854 **SEE ALSO**

15855 **<locale.h>**, **<signal.h>**, **<stddef.h>**, **<sys/types.h>**

15856 XSH Section 2.2 (on page 496), *asctime*(), *clock*(), *clock_getcpuclockid*(), *clock_getres*(),
15857 *clock_nanosleep*(), *ctime*(), *difftime*(), *futimens*(), *getdate*(), *gmtime*(), *localtime*(), *mktime*(),
15858 *mq_receive*(), *mq_send*(), *nanosleep*(), *pthread_getcpuclockid*(), *pthread_mutex_clocklock*(),

15859 *pthread_rwlock_clockrdlock*( ), *pthread_rwlock_clockwrlock*( ), *sem_clockwait*( ), *strftime*( ), *strptime*( ),
15860 *sysconf*( ), *time*( ), *timer_create*( ), *timer_delete*( ), *timer_getoverrun*( ), *timespec_get*( ), *tzset*( )

15861 **CHANGE HISTORY**

15862 First released in Issue 1. Derived from Issue 1 of the SVID.

15863 **Issue 5**

15864 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
15865 Threads Extension.

15866 **Issue 6**

15867 The Open Group Corrigendum U035/6 is applied. In the DESCRIPTION, the types **clockid_t**
15868 and **timer_t** have been described.

15869 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

15870 • The POSIX timer-related functions are marked as part of the Timers option.

15871 The symbolic name CLK_TCK is removed. Application usage is added describing how its
15872 equivalent functionality can be obtained using *sysconf*( ).

15873 The *clock_getcpuclockid*( ) function and manifest constants CLOCK_PROCESS_CPUTIME_ID and
15874 CLOCK_THREAD_CPUTIME_ID are added for alignment with IEEE Std 1003.1d-1999.

15875 The manifest constant CLOCK_MONOTONIC and the *clock_nanosleep*( ) function are added for
15876 alignment with IEEE Std 1003.1j-2000.

15877 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

15878 • The range for seconds is changed from [0,61] to [0,60].

15879 • The **restrict** keyword is added to the prototypes for *asctime_r*( ), *gmtime_r*( ), *localtime_r*( ),
15880 *strftime*( ), *strptime*( ), *timer_create*( ), and *timer_settime*( ).

15881 IEEE PASC Interpretation 1003.1 #84 is applied adding the statement that symbols from the
15882 **<signal.h>** header may be made visible when the **<time.h>** header is included.

15883 Extensions beyond the ISO C standard are marked.

15884 **Issue 7**

15885 Austin Group Interpretation 1003.1-2001 #111 is applied.

15886 SD5-XBD-ERN-74 is applied.

15887 The *strftime_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API
15888 Set Part 4.

15889 Functionality relating to the Timers option is moved to the Base.

15890 This reference page is clarified with respect to macros and symbolic constants, and declarations
15891 for the **locale_t** and **pid_t** types and the **sigevent** structure are added.

15892 The description of the *getdate_err* value is expanded.

15893 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0076 [212] and XBD/TC1-2008/0077
15894 [212] are applied.

15895 **Issue 8**

15896 Austin Group Defect 1253 is applied, changing ``Daylight Savings'' to ``Daylight Saving''.

15897 Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899: 2018 standard.

15898 Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.

15899 Austin Group Defect 1410 is applied, removing the *asctime_r*( ) and *ctime_r*( ) functions.

15900 Austin Group Defect 1533 is applied, adding *tm_gmtoff* and *tm_zone* to the **tm** structure.

15901 Austin Group Defect 1597 is applied, changing *clock*( ) to *clock**( ) in the descriptions of
15902 CLOCK_PROCESS_CPUTIME_ID and CLOCK_THREAD_CPUTIME_ID.

15903 Austin Group Defect 1614 is applied, clarifying the requirements for the *tm_isdst* member of the
15904 **tm** structure.

NAME

uchar.h — Unicode character handling

SYNOPSIS

```
#include <uchar.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The **<uchar.h>** header shall define the following types:

mbstate_t As described in **<wchar.h>**.

size_t As described in **<stddef.h>**.

char16_t The same type as **uint_least16_t**, described in **<stdint.h>**.

char32_t The same type as **uint_least32_t**, described in **<stdint.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
size_t  c16rtomb(char *restrict, char16_t, mbstate_t *restrict);
size_t  c32rtomb(char *restrict, char32_t, mbstate_t *restrict);
size_t  mbrtoc16(char16_t *restrict, const char *restrict, size_t,
            mbstate_t *restrict);
size_t  mbrtoc32(char32_t *restrict, const char *restrict, size_t,
            mbstate_t *restrict);
```

CX Inclusion of the **<uchar.h>** header may make visible all symbols from the headers **<stddef.h>**, **<stdint.h>**, and **<wchar.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<stddef.h>, **<stdint.h>**, **<wchar.h>**

XSH *c16rtomb*( ), *mbrtoc16*( )

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

unistd.h — standard symbolic constants and types

SYNOPSIS

```
#include <unistd.h>
```

DESCRIPTION

The **<unistd.h>** header defines miscellaneous symbolic constants and types, and declares miscellaneous functions. The actual values of the constants are unspecified except as shown. The contents of this header are shown below.

Version Test Macros

The **<unistd.h>** header shall define the following symbolic constants. The values shall be suitable for use in **#if** preprocessing directives.

_POSIX_VERSION
: Integer value indicating version of this standard (C-language binding) to which the implementation conforms. For implementations conforming to POSIX.1-2024, the value shall be 202405L.

_POSIX2_VERSION
: Integer value indicating version of the Shell and Utilities volume of POSIX.1 to which the implementation conforms. For implementations conforming to POSIX.1-2024, the value shall be 202405L. For profile implementations that define _POSIX_SUBPROFILE (see Section 2.1.5.1) in **<unistd.h>**, _POSIX2_VERSION may be left undefined or be defined with the value –1 to indicate that the Shell and Utilities volume of POSIX.1 is not supported. In this case, a call to *sysconf* (_SC_2_VERSION) shall return either 202405L or –1 indicating that the Shell and Utilities volume of POSIX.1 is or is not, respectively, supported at runtime.

The **<unistd.h>** header shall define the following symbolic constant only if the implementation supports the XSI option; see Section 2.1.4 (on page 19). If defined, its value shall be suitable for use in **#if** preprocessing directives.

XSI _XOPEN_VERSION
: Integer value indicating version of the X/Open Portability Guide to which the implementation conforms. The value shall be 800.

Constants for Options and Option Groups

The following symbolic constants, if defined in **<unistd.h>**, shall have a value of –1, 0, or greater, unless otherwise specified below. For profile implementations that define _POSIX_SUBPROFILE (see Section 2.1.5.1) in **<unistd.h>**, constants described below as always having a value greater than zero need not be defined and, if defined, may have a value of –1, 0, or greater. The values shall be suitable for use in **#if** preprocessing directives.

If a symbolic constant is not defined or is defined with the value –1, the option is not supported for compilation. If it is defined with a value greater than zero, the option shall always be supported when the application is executed. If it is defined with the value zero, the option shall be supported for compilation and might or might not be supported at runtime. See Section 2.1.6 (on page 25) for further information about the conformance requirements of these three categories of support.

ADV _POSIX_ADVISORY_INFO
: The implementation supports the Advisory Information option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

15984 _POSIX_ASYNCHRONOUS_IO
15985 The implementation supports asynchronous input and output. This symbol shall always be
15986 set to the value 202405L.

15987 _POSIX_BARRIERS
15988 The implementation supports barriers. This symbol shall always be set to the value
15989 202405L.

15990 _POSIX_CHOWN_RESTRICTED
15991 The use of *chown*( ) and *fchown*( ) is restricted to a process with appropriate privileges, and
15992 to changing the group ID of a file only to the effective group ID of the process or to one of
15993 its supplementary group IDs. This symbol shall be defined with a value other than –1.

15994 _POSIX_CLOCK_SELECTION
15995 The implementation supports clock selection. This symbol shall always be set to the value
15996 202405L.

15997 CPT _POSIX_CPUTIME
15998 The implementation supports the Process CPU-Time Clocks option. If this symbol is
15999 defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol
16000 reported by *sysconf*( ) shall either be –1 or 202405L.

16001 DC _POSIX_DEVICE_CONTROL
16002 The implementation supports the device control option. If this symbol is defined in
16003 **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by
16004 *sysconf*( ) shall either be –1 or 202405L.

16005 FSC _POSIX_FSYNC
16006 The implementation supports the File Synchronization option. If this symbol is defined in
16007 **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by
16008 *sysconf*( ) shall either be –1 or 202405L.

16009 IP6 _POSIX_IPV6
16010 The implementation supports the IPv6 option. If this symbol is defined in **<unistd.h>**, it
16011 shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall
16012 either be –1 or 202405L.

16013 _POSIX_JOB_CONTROL
16014 The implementation supports job control. This symbol shall always be set to a value greater
16015 than zero.

16016 _POSIX_MAPPED_FILES
16017 The implementation supports memory mapped Files. This symbol shall always be set to the
16018 value 202405L.

16019 ML _POSIX_MEMLOCK
16020 The implementation supports the Process Memory Locking option. If this symbol is defined
16021 in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported
16022 by *sysconf*( ) shall either be –1 or 202405L.

16023 MLR _POSIX_MEMLOCK_RANGE
16024 The implementation supports the Range Memory Locking option. If this symbol is defined
16025 in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported
16026 by *sysconf*( ) shall either be –1 or 202405L.

16027 _POSIX_MEMORY_PROTECTION
16028 The implementation supports memory protection. This symbol shall always be set to the
16029 value 202405L.

MSG _POSIX_MESSAGE_PASSING
The implementation supports the Message Passing option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

_POSIX_MONOTONIC_CLOCK
The implementation supports a monotonic clock. This symbol shall always be set to the value 202405L.

_POSIX_NO_TRUNC
Pathname components longer than {NAME_MAX} generate an error. This symbol shall be defined with a value other than –1.

PIO _POSIX_PRIORITIZED_IO
The implementation supports the Prioritized Input and Output option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

PS _POSIX_PRIORITY_SCHEDULING
The implementation supports the Process Scheduling option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

RS _POSIX_RAW_SOCKETS
The implementation supports the Raw Sockets option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

_POSIX_READER_WRITER_LOCKS
The implementation supports read-write locks. This symbol shall always be set to the value 202405L.

_POSIX_REALTIME_SIGNALS
The implementation supports realtime signals. This symbol shall always be set to the value 202405L.

_POSIX_REGEXP
The implementation supports the Regular Expression Handling option. This symbol shall always be set to a value greater than zero.

_POSIX_SAVED_IDS
Each process has a saved set-user-ID and a saved set-group-ID. This symbol shall always be set to a value greater than zero.

_POSIX_SEMAPHORES
The implementation supports semaphores. This symbol shall always be set to the value 202405L.

SHM _POSIX_SHARED_MEMORY_OBJECTS
The implementation supports the Shared Memory Objects option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

_POSIX_SHELL
The implementation supports the POSIX shell. This symbol shall always be set to a value greater than zero.

SPN _POSIX_SPAWN
The implementation supports the Spawn option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

_POSIX_SPIN_LOCKS
The implementation supports spin locks. This symbol shall always be set to the value 202405L.

SS _POSIX_SPORADIC_SERVER
The implementation supports the Process Sporadic Server option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

SIO _POSIX_SYNCHRONIZED_IO
The implementation supports the Synchronized Input and Output option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

TSA _POSIX_THREAD_ATTR_STACKADDR
The implementation supports the Thread Stack Address Attribute option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

TSS _POSIX_THREAD_ATTR_STACKSIZE
The implementation supports the Thread Stack Size Attribute option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

TCT _POSIX_THREAD_CPUTIME
The implementation supports the Thread CPU-Time Clocks option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

TPI _POSIX_THREAD_PRIO_INHERIT
The implementation supports the Non-Robust Mutex Priority Inheritance option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

TPP _POSIX_THREAD_PRIO_PROTECT
The implementation supports the Non-Robust Mutex Priority Protection option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

TPS _POSIX_THREAD_PRIORITY_SCHEDULING
The implementation supports the Thread Execution Scheduling option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

TSH _POSIX_THREAD_PROCESS_SHARED
The implementation supports the Thread Process-Shared Synchronization option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

RPI _POSIX_THREAD_ROBUST_PRIO_INHERIT
The implementation supports the Robust Mutex Priority Inheritance option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol

16120 reported by *sysconf*( ) shall either be –1 or 202405L.

16121 RPP _POSIX_THREAD_ROBUST_PRIO_PROTECT
16122 The implementation supports the Robust Mutex Priority Protection option. If this symbol is
16123 defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol
16124 reported by *sysconf*( ) shall either be –1 or 202405L.

16125 _POSIX_THREAD_SAFE_FUNCTIONS
16126 The implementation supports thread-safe functions. This symbol shall always be set to the
16127 value 202405L.

16128 TSP _POSIX_THREAD_SPORADIC_SERVER
16129 The implementation supports the Thread Sporadic Server option. If this symbol is defined
16130 in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported
16131 by *sysconf*( ) shall either be –1 or 202405L.

16132 _POSIX_THREADS
16133 The implementation supports threads. This symbol shall always be set to the value
16134 202405L.

16135 _POSIX_TIMEOUTS
16136 The implementation supports timeouts. This symbol shall always be set to the value
16137 202405L.

16138 _POSIX_TIMERS
16139 The implementation supports timers. This symbol shall always be set to the value 202405L.

16140 TYM _POSIX_TYPED_MEMORY_OBJECTS
16141 The implementation supports the Typed Memory Objects option. If this symbol is defined
16142 in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported
16143 by *sysconf*( ) shall either be –1 or 202405L.

16144 OB _POSIX_V7_ILP32_OFF32
16145 The implementation provides a C-language compilation environment with 32-bit **int**, **long**,
16146 **off_t**, and all pointer types.

16147 OB _POSIX_V7_ILP32_OFFBIG
16148 The implementation provides a C-language compilation environment with 32-bit **int**, **long**,
16149 and all pointer types, and an **off_t** type using at least 64 bits.

16150 OB _POSIX_V7_LP64_OFF64
16151 The implementation provides a C-language compilation environment with a 32-bit **int** type
16152 and 64-bit **long**, **off_t**, and all pointer types.

16153 OB _POSIX_V7_LPBIG_OFFBIG
16154 The implementation provides a C-language compilation environment with an **int** type
16155 using at least 32 bits and **long**, **off_t**, and all pointer types using at least 64 bits.

16156 _POSIX_V8_ILP32_OFF32
16157 The implementation provides a C-language compilation environment with 32-bit **int**, **long**,
16158 **off_t**, and all pointer types.

16159 _POSIX_V8_ILP32_OFFBIG
16160 The implementation provides a C-language compilation environment with 32-bit **int**, **long**,
16161 and all pointer types, and an **off_t** type using at least 64 bits.

16162 _POSIX_V8_LP64_OFF64
16163 The implementation provides a C-language compilation environment with a 32-bit **int** type
16164 and 64-bit **long**, **off_t**, and all pointer types.

_POSIX_V8_LPBIG_OFFBIG
The implementation provides a C-language compilation environment with an **int** type using at least 32 bits and **long**, **off_t**, and all pointer types using at least 64 bits.

_POSIX2_C_BIND
The implementation supports the C-Language Binding option. This symbol shall always have the value 202405L.

CD _POSIX2_C_DEV
The implementation supports the C-Language Development Utilities option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

_POSIX2_CHAR_TERM
The implementation supports the Terminal Characteristics option. The value of this symbol reported by *sysconf*( ) shall either be –1 or a value greater than zero.

FR _POSIX2_FORT_RUN
The implementation supports the FORTRAN Runtime Utilities option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

_POSIX2_LOCALEDEF
The implementation supports the creation of locales by the *localedef* utility. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

SD _POSIX2_SW_DEV
The implementation supports the Software Development Utilities option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

UP _POSIX2_UPE
The implementation supports the User Portability Utilities option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by *sysconf*( ) shall either be –1 or 202405L.

XSI _XOPEN_CRYPT
The implementation supports the X/Open Encryption Option Group.

_XOPEN_ENH_I18N
The implementation supports the Issue 4, Version 2 Enhanced Internationalization Option Group. This symbol shall always be set to a value other than –1.

_XOPEN_REALTIME
The implementation supports the X/Open Realtime Option Group.

_XOPEN_REALTIME_THREADS
The implementation supports the X/Open Realtime Threads Option Group.

_XOPEN_SHM
The implementation supports the Issue 4, Version 2 Shared Memory Option Group. This symbol shall always be set to a value other than –1.

_XOPEN_UNIX
The implementation supports the XSI option.

16208 UU _XOPEN_UUCP
16209 The implementation supports the UUCP Utilities option. If this symbol is defined in
16210 **<unistd.h>**, it shall be defined to be –1, 0, or 202405L. The value of this symbol reported by
16211 *sysconf*( ) shall be either –1 or 202405L.

16212 **Execution-Time Symbolic Constants**

16213 If any of the following symbolic constants are not defined in the **<unistd.h>** header, the value
16214 shall vary depending on the file to which it is applied. If defined, they shall have values suitable
16215 for use in **#if** preprocessing directives.

16216 If any of the following symbolic constants are defined to have value –1 in the **<unistd.h>** header,
16217 the implementation shall not provide the option on any file; if any are defined to have a value
16218 other than –1 in the **<unistd.h>** header, the implementation shall provide the option on all
16219 applicable files.

16220 All of the following values, whether defined as symbolic constants in **<unistd.h>** or not, may be
16221 queried with respect to a specific file using the *pathconf*( ) or *fpathconf*( ) functions:

16222 _POSIX_ASYNC_IO
16223 Asynchronous input or output operations may be performed for the associated file.

16224 _POSIX_FALLOC
16225 The *posix_fallocate*( ) function is supported by the associated file.

16226 _POSIX_PRIO_IO
16227 Prioritized input or output operations may be performed for the associated file.

16228 _POSIX_SYNC_IO
16229 Synchronized input or output operations may be performed for the associated file.

16230 If the following symbolic constants are defined in the **<unistd.h>** header, they apply to files and
16231 all paths in all file systems on the implementation:

16232 _POSIX_TIMESTAMP_RESOLUTION
16233 The resolution in nanoseconds for all file timestamps.

16234 _POSIX2_SYMLINKS
16235 Symbolic links can be created.

16236 **Constants for Functions**

16237 The **<unistd.h>** header shall define NULL as described in **<stddef.h>**.

16238 The **<unistd.h>** header shall define the symbolic constants O_CLOEXEC and O_CLOFORK as
16239 described in **<fcntl.h>**.

16240 The **<unistd.h>** header shall define the following symbolic constants for use with the *access*( )
16241 function. The values shall be suitable for use in **#if** preprocessing directives.

16242 F_OK Test for existence of file.

16243 R_OK Test for read permission.

16244 W_OK Test for write permission.

16245 X_OK Test for execute (search) permission.

16246 The constants F_OK, R_OK, W_OK, and X_OK and the expressions *R_OK*|*W_OK*, *R_OK*|*X_OK*,
16247 and *R_OK*|*W_OK*|*X_OK* shall all have distinct values.

The **<unistd.h>** header shall define the following symbolic constants for the *confstr*( ) function:

_CS_PATH
: This is the value for the *PATH* environment variable that finds all of the standard utilities that are provided in a manner accessible via the *exec* family of functions.

_CS_POSIX_V8_ILP32_OFF32_CFLAGS
: If *sysconf*(_SC_V8_ILP32_OFF32) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of initial options to be given to the *c17* utility to build an application using a programming model with 32-bit **int**, **long**, **off_t**, and all pointer types.

_CS_POSIX_V8_ILP32_OFF32_LDFLAGS
: If *sysconf*(_SC_V8_ILP32_OFF32) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of final options to be given to the *c17* utility to build an application using a programming model with 32-bit **int**, **long**, **off_t**, and all pointer types.

_CS_POSIX_V8_ILP32_OFF32_LIBS
: If *sysconf*(_SC_V8_ILP32_OFF32) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of libraries to be given to the *c17* utility to build an application using a programming model with 32-bit **int**, **long**, **off_t**, and all pointer types.

_CS_POSIX_V8_ILP32_OFFBIG_CFLAGS
: If *sysconf*(_SC_V8_ILP32_OFFBIG) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of initial options to be given to the *c17* utility to build an application using a programming model with 32-bit **int**, **long**, and all pointer types, and an **off_t** type using at least 64 bits.

_CS_POSIX_V8_ILP32_OFFBIG_LDFLAGS
: If *sysconf*(_SC_V8_ILP32_OFFBIG) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of final options to be given to the *c17* utility to build an application using a programming model with 32-bit **int**, **long**, and all pointer types, and an **off_t** type using at least 64 bits.

_CS_POSIX_V8_ILP32_OFFBIG_LIBS
: If *sysconf*(_SC_V8_ILP32_OFFBIG) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of libraries to be given to the *c17* utility to build an application using a programming model with 32-bit **int**, **long**, and all pointer types, and an **off_t** type using at least 64 bits.

_CS_POSIX_V8_LP64_OFF64_CFLAGS
: If *sysconf*(_SC_V8_LP64_OFF64) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of initial options to be given to the *c17* utility to build an application using a programming model with a 32-bit **int** type and 64-bit **long**, **off_t**, and all pointer types.

_CS_POSIX_V8_LP64_OFF64_LDFLAGS
: If *sysconf*(_SC_V8_LP64_OFF64) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of final options to be given to the *c17* utility to build an application using a programming model with a 32-bit **int** type and 64-bit **long**, **off_t**, and all pointer types.

_CS_POSIX_V8_LP64_OFF64_LIBS
: If *sysconf*(_SC_V8_LP64_OFF64) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of libraries to be given to the *c17* utility to build an application using a programming model with a 32-bit **int** type and 64-bit **long**, **off_t**, and all pointer types.

16294 _CS_POSIX_V8_LPBIG_OFFBIG_CFLAGS
16295 If *sysconf*(_SC_V8_LPBIG_OFFBIG) returns –1, the meaning of this value is unspecified.
16296 Otherwise, this value is the set of initial options to be given to the *c17* utility to build an
16297 application using a programming model with an **int** type using at least 32 bits and **long**,
16298 **off_t**, and all pointer types using at least 64 bits.

16299 _CS_POSIX_V8_LPBIG_OFFBIG_LDFLAGS
16300 If *sysconf*(_SC_V8_LPBIG_OFFBIG) returns –1, the meaning of this value is unspecified.
16301 Otherwise, this value is the set of final options to be given to the *c17* utility to build an
16302 application using a programming model with an **int** type using at least 32 bits and **long**,
16303 **off_t**, and all pointer types using at least 64 bits.

16304 _CS_POSIX_V8_LPBIG_OFFBIG_LIBS
16305 If *sysconf*(_SC_V8_LPBIG_OFFBIG) returns –1, the meaning of this value is unspecified.
16306 Otherwise, this value is the set of libraries to be given to the *c17* utility to build an
16307 application using a programming model with an **int** type using at least 32 bits and **long**,
16308 **off_t**, and all pointer types using at least 64 bits.

16309 _CS_POSIX_V8_THREADS_CFLAGS
16310 This value is the set of initial options to be given to the *c17* utility to build a multi-threaded
16311 application. These flags are in addition to those associated with any of the other
16312 _CS_POSIX_V8_*_CFLAGS values used to specify particular type size programming
16313 environments.

16314 _CS_POSIX_V8_THREADS_LDFLAGS
16315 This value is the set of final options to be given to the *c17* utility to build a multi-threaded
16316 application. These flags are in addition to those associated with any of the other
16317 _CS_POSIX_V8_*_LDFLAGS values used to specify particular type size programming
16318 environments.

16319 _CS_POSIX_V8_WIDTH_RESTRICTED_ENVS
16320 This value is a <newline>-separated list of names of programming environments supported
16321 by the implementation in which the widths of the **blksize_t**, **cc_t**, **mode_t**, **nfds_t**, **pid_t**,
16322 **ptrdiff_t**, **size_t**, **speed_t**, **ssize_t**, **suseconds_t**, **tcflag_t**, **wchar_t**, and **wint_t** types are no
16323 greater than the width of type **long**. The format of each name shall be suitable for use with
16324 the *getconf* **–v** option.

16325 _CS_V8_ENV
16326 This is the value that provides the environment variable information (other than that
16327 provided by _CS_PATH) that is required by the implementation to create a conforming
16328 environment, as described in the implementation's conformance documentation.

16329 OB The following symbolic constants are reserved for compatibility with Issue 7:

16330 _CS_POSIX_V7_ILP32_OFF32_CFLAGS
16331 _CS_POSIX_V7_ILP32_OFF32_LDFLAGS
16332 _CS_POSIX_V7_ILP32_OFF32_LIBS
16333 _CS_POSIX_V7_ILP32_OFFBIG_CFLAGS
16334 _CS_POSIX_V7_ILP32_OFFBIG_LDFLAGS
16335 _CS_POSIX_V7_ILP32_OFFBIG_LIBS
16336 _CS_POSIX_V7_LP64_OFF64_CFLAGS
16337 _CS_POSIX_V7_LP64_OFF64_LDFLAGS
16338 _CS_POSIX_V7_LP64_OFF64_LIBS
16339 _CS_POSIX_V7_LPBIG_OFFBIG_CFLAGS
16340 _CS_POSIX_V7_LPBIG_OFFBIG_LDFLAGS
16341 _CS_POSIX_V7_LPBIG_OFFBIG_LIBS

16342 _CS_POSIX_V7_THREADS_CFLAGS
16343 _CS_POSIX_V7_THREADS_LDFLAGS
16344 _CS_POSIX_V7_WIDTH_RESTRICTED_ENVS
16345 _CS_V7_ENV

16346 The implementation may define additional symbolic constants beginning with _CS_ for use by
16347 *confstr*( ).

16348 The **<unistd.h>** header shall define SEEK_CUR, SEEK_END, and SEEK_SET as described in
16349 **<stdio.h>**.

16350 Additionally, it shall define the following macros which shall expand to integer constant
16351 expressions with values that are distinct from each other and from SEEK_CUR, SEEK_END, and
16352 SEEK_SET:

16353 SEEK_HOLE Seek forwards from offset relative to start-of-file for a position within a hole.

16354 SEEK_DATA Seek forwards from offset relative to start-of-file for a position not within a
16355 hole

16356 XSI The **<unistd.h>** header shall define the following symbolic constants as possible values for the
16357 *function* argument to the *lockf*( ) function:

16358 F_LOCK Lock a section for exclusive use.

16359 F_TEST Test section for locks by other processes.

16360 F_TLOCK Test and lock a section for exclusive use.

16361 F_ULOCK Unlock locked sections.

16362 The **<unistd.h>** header shall define the following symbolic constants for *pathconf*( ):

16363 _PC_2_SYMLINKS
16364 _PC_ALLOC_SIZE_MIN
16365 _PC_ASYNC_IO
16366 _PC_CHOWN_RESTRICTED
16367 _PC_FALLOC
16368 _PC_FILESIZEBITS
16369 _PC_LINK_MAX
16370 _PC_MAX_CANON
16371 _PC_MAX_INPUT
16372 _PC_NAME_MAX
16373 _PC_NO_TRUNC
16374 _PC_PATH_MAX
16375 _PC_PIPE_BUF
16376 _PC_PRIO_IO
16377 _PC_REC_INCR_XFER_SIZE
16378 _PC_REC_MAX_XFER_SIZE
16379 _PC_REC_MIN_XFER_SIZE
16380 _PC_REC_XFER_ALIGN
16381 _PC_SYMLINK_MAX
16382 _PC_SYNC_IO
16383 _PC_TEXTDOMAIN_MAX
16384 _PC_TIMESTAMP_RESOLUTION
16385 _PC_VDISABLE

The implementation may define additional symbolic constants beginning with _PC_ for use by *pathconf*( ).

The **<unistd.h>** header shall define the following symbolic constants for *sysconf*( ):

_SC_2_C_BIND
_SC_2_C_DEV
_SC_2_CHAR_TERM
_SC_2_FORT_RUN
_SC_2_LOCALEDEF
_SC_2_SW_DEV
_SC_2_UPE
_SC_2_VERSION
_SC_ADVISORY_INFO
_SC_AIO_LISTIO_MAX
_SC_AIO_MAX
_SC_AIO_PRIO_DELTA_MAX
_SC_ARG_MAX
_SC_ASYNCHRONOUS_IO
_SC_ATEXIT_MAX
_SC_BARRIERS
_SC_BC_BASE_MAX
_SC_BC_DIM_MAX
_SC_BC_SCALE_MAX
_SC_BC_STRING_MAX
_SC_CHILD_MAX
_SC_CLK_TCK
_SC_CLOCK_SELECTION
_SC_COLL_WEIGHTS_MAX
_SC_CPUTIME
_SC_DELAYTIMER_MAX
_SC_DEVICE_CONTROL
_SC_EXPR_NEST_MAX
_SC_FSYNC
_SC_GETGR_R_SIZE_MAX
_SC_GETPW_R_SIZE_MAX
_SC_HOST_NAME_MAX
_SC_IOV_MAX
_SC_IPV6
_SC_JOB_CONTROL
_SC_LINE_MAX
_SC_LOGIN_NAME_MAX
_SC_MAPPED_FILES
_SC_MEMLOCK
_SC_MEMLOCK_RANGE
_SC_MEMORY_PROTECTION
_SC_MESSAGE_PASSING
_SC_MONOTONIC_CLOCK
_SC_MQ_OPEN_MAX
_SC_MQ_PRIO_MAX
_SC_NGROUPS_MAX
_SC_NPROCESSORS_CONF
_SC_NPROCESSORS_ONLN

16437 _SC_NSIG
16438 _SC_OPEN_MAX
16439 _SC_PAGE_SIZE
16440 _SC_PAGESIZE
16441 _SC_PRIORITIZED_IO
16442 _SC_PRIORITY_SCHEDULING
16443 _SC_RAW_SOCKETS
16444 _SC_RE_DUP_MAX
16445 _SC_READER_WRITER_LOCKS
16446 _SC_REALTIME_SIGNALS
16447 _SC_REGEXP
16448 _SC_RTSIG_MAX
16449 _SC_SAVED_IDS
16450 _SC_SEM_NSEMS_MAX
16451 _SC_SEM_VALUE_MAX
16452 _SC_SEMAPHORES
16453 _SC_SHARED_MEMORY_OBJECTS
16454 _SC_SHELL
16455 _SC_SIGQUEUE_MAX
16456 _SC_SPAWN
16457 _SC_SPIN_LOCKS
16458 _SC_SPORADIC_SERVER
16459 _SC_SS_REPL_MAX
16460 _SC_STREAM_MAX
16461 _SC_SYMLOOP_MAX
16462 _SC_SYNCHRONIZED_IO
16463 _SC_THREAD_ATTR_STACKADDR
16464 _SC_THREAD_ATTR_STACKSIZE
16465 _SC_THREAD_CPUTIME
16466 _SC_THREAD_DESTRUCTOR_ITERATIONS
16467 _SC_THREAD_KEYS_MAX
16468 _SC_THREAD_PRIO_INHERIT
16469 _SC_THREAD_PRIO_PROTECT
16470 _SC_THREAD_PRIORITY_SCHEDULING
16471 _SC_THREAD_PROCESS_SHARED
16472 _SC_THREAD_ROBUST_PRIO_INHERIT
16473 _SC_THREAD_ROBUST_PRIO_PROTECT
16474 _SC_THREAD_SAFE_FUNCTIONS
16475 _SC_THREAD_SPORADIC_SERVER
16476 _SC_THREAD_STACK_MIN
16477 _SC_THREAD_THREADS_MAX
16478 _SC_THREADS
16479 _SC_TIMEOUTS
16480 _SC_TIMER_MAX
16481 _SC_TIMERS
16482 _SC_TTY_NAME_MAX
16483 _SC_TYPED_MEMORY_OBJECTS
16484 _SC_TZNAME_MAX
16485 _SC_V8_ILP32_OFF32
16486 _SC_V8_ILP32_OFFBIG
16487 _SC_V8_LP64_OFF64
16488 _SC_V8_LPBIG_OFFBIG

OB _SC_V7_ILP32_OFF32
_SC_V7_ILP32_OFFBIG
_SC_V7_LP64_OFF64
_SC_V7_LPBIG_OFFBIG
_SC_VERSION
_SC_XOPEN_CRYPT
_SC_XOPEN_ENH_I18N
_SC_XOPEN_REALTIME
_SC_XOPEN_REALTIME_THREADS
_SC_XOPEN_SHM
_SC_XOPEN_UNIX
_SC_XOPEN_UUCP
_SC_XOPEN_VERSION

The two constants _SC_PAGESIZE and _SC_PAGE_SIZE may be defined to have the same value.

The implementation may define additional symbolic constants beginning with _SC_ for use by *sysconf*( ).

The **<unistd.h>** header shall define the following symbolic constants for file streams:

STDERR_FILENO File number of *stderr*; 2.

STDIN_FILENO File number of *stdin*; 0.

STDOUT_FILENO File number of *stdout*; 1.

The **<unistd.h>** header shall define the following symbolic constant for terminal special character handling:

_POSIX_VDISABLE This symbol shall be defined to be the value of a character that shall disable terminal special character handling as described in Section 11.2.6 (on page 212). This symbol shall always be set to a value other than –1.

The **<unistd.h>** header shall define the following symbolic constant as a value for the flag used by *posix_close*( ):

POSIX_CLOSE_RESTART
Allows restarts if a signal interrupts a close operation. This constant shall not be 0 unless *posix_close*( ) never returns –1 with *errno* set to [EINTR].

Type Definitions

The **<unistd.h>** header shall define the **size_t**, **ssize_t**, **uid_t**, **gid_t**, **off_t**, and **pid_t** types as described in **<sys/types.h>**.

The **<unistd.h>** header shall define the **intptr_t** type as described in **<stdint.h>**.

Declarations

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int          access(const char *, int);
unsigned     alarm(unsigned);
int          chdir(const char *);
int          chown(const char *, uid_t, gid_t);
int          close(int);
size_t       confstr(int, char *, size_t);
```



```
16532 XSI       char         *crypt(const char *, const char *);
16533           int           dup(int);
16534           int           dup2(int, int);
16535           int           dup3(int, int, int);
16536           _Noreturn void
16537                         _exit(int);
16538 OB XSI    void          encrypt(char [64], int);
16539           int           execl(const char *, const char *, ...);
16540           int           execle(const char *, const char *, ...);
16541           int           execlp(const char *, const char *, ...);
16542           int           execv(const char *, char *const []);
16543           int           execve(const char *, char *const [], char *const []);
16544           int           execvp(const char *, char *const []);
16545           int           faccessat(int, const char *, int, int);
16546           int           fchdir(int);
16547           int           fchown(int, uid_t, gid_t);
16548           int           fchownat(int, const char *, uid_t, gid_t, int);
16549 SIO       int           fdatasync(int);
16550           int           fexecve(int, char *const [], char *const []);
16551           pid_t         _Fork(void);
16552           pid_t         fork(void);
16553           long          fpathconf(int, int);
16554 FSC       int           fsync(int);
16555           int           ftruncate(int, off_t);
16556           char         *getcwd(char *, size_t);
16557           gid_t         getegid(void);
16558           int           getentropy(void *, size_t);
16559           uid_t         geteuid(void);
16560           gid_t         getgid(void);
16561           int           getgroups(int, gid_t []);
16562 XSI       long          gethostid(void);
16563           int           gethostname(char *, size_t);
16564           char         *getlogin(void);
16565           int           getlogin_r(char *, size_t);
16566           int           getopt(int, char * const [], const char *);
16567           pid_t         getpgid(pid_t);
16568           pid_t         getpgrp(void);
16569           pid_t         getpid(void);
16570           pid_t         getppid(void);
16571 XSI       int           getresgid(gid_t *restrict, gid_t *restrict,
16572                             gid_t *restrict);
16573           int           getresuid(uid_t *restrict, uid_t *restrict,
16574                             uid_t *restrict);
16575           pid_t         getsid(pid_t);
16576           uid_t         getuid(void);
16577           int           isatty(int);
16578           int           lchown(const char *, uid_t, gid_t);
16579           int           link(const char *, const char *);
16580           int           linkat(int, const char *, int, const char *, int);
16581 XSI       int           lockf(int, int, off_t);
16582           off_t         lseek(int, off_t, int);
```



```
16583 XSI       int             nice(int);
16584           long            pathconf(const char *, int);
16585           int             pause(void);
16586           int             pipe(int [2]);
16587           int             pipe2(int [2], int);
16588           int             posix_close(int, int);
16589           ssize_t         pread(int, void *, size_t, off_t);
16590           ssize_t         pwrite(int, const void *, size_t, off_t);
16591           ssize_t         read(int, void *, size_t);
16592           ssize_t         readlink(const char *restrict, char *restrict, size_t);
16593           ssize_t         readlinkat(int, const char *restrict, char *restrict,
16594                               size_t);
16595           int             rmdir(const char *);
16596           int             setegid(gid_t);
16597           int             seteuid(uid_t);
16598           int             setgid(gid_t);
16599           int             setpgid(pid_t, pid_t);
16600 XSI       int             setregid(gid_t, gid_t);
16601           int             setresgid(gid_t, gid_t, gid_t);
16602           int             setresuid(uid_t, uid_t, uid_t);
16603           int             setreuid(uid_t, uid_t);
16604           pid_t           setsid(void);
16605           int             setuid(uid_t);
16606           unsigned        sleep(unsigned);
16607 XSI       void            swab(const void *restrict, void *restrict, ssize_t);
16608           int             symlink(const char *, const char *);
16609           int             symlinkat(const char *, int, const char *);
16610 XSI       void            sync(void);
16611           long            sysconf(int);
16612           pid_t           tcgetpgrp(int);
16613           int             tcsetpgrp(int, pid_t);
16614           int             truncate(const char *, off_t);
16615           char           *ttyname(int);
16616           int             ttyname_r(int, char *, size_t);
16617           int             unlink(const char *);
16618           int             unlinkat(int, const char *, int);
16619           ssize_t         write(int, const void *, size_t);
```

16620 The **<unistd.h>** header shall declare the following external variables:

```
16621           extern char  *optarg;
16622           extern int    opterr, optind, optopt;
```

16623 Inclusion of the **<unistd.h>** header may make visible all symbols from the headers **<fcntl.h>**,
16624 **<stddef.h>**, **<stdint.h>**, and **<stdio.h>**.

16625 **APPLICATION USAGE**

16626 POSIX.1-2024 only describes the behavior of systems that claim conformance to it. However,
16627 application developers who want to write applications that adapt to other versions of this
16628 standard (or to systems that do not conform to any POSIX standard) may find it useful to code
16629 them so as to conditionally compile different code depending on the value of
16630 _POSIX_VERSION, for example:

```
16631 #if _POSIX_VERSION >= 200112L
16632 /* Use the newer function that copes with large files. */
16633 off_t pos=ftello(fp);
16634 #else
16635 /* Either this is an old version of POSIX, or _POSIX_VERSION is
16636    not even defined, so use the traditional function. */
16637 long pos=ftell(fp);
16638 #endif
```

16639 Earlier versions of POSIX.1-2024 and of the Single UNIX Specification can be identified by the
16640 following macros:

16641 POSIX.1-1988 standard
16642 _POSIX_VERSION == 198808L

16643 POSIX.1-1990 standard
16644 _POSIX_VERSION == 199009L

16645 POSIX.1-1996 standard
16646 _POSIX_VERSION == 199506L

16647 Single UNIX Specification, Version 1
16648 _XOPEN_UNIX and _XOPEN_VERSION == 4

16649 Single UNIX Specification, Version 2
16650 _XOPEN_UNIX and _XOPEN_VERSION == 500

16651 POSIX.1-2001 and Single UNIX Specification, Version 3
16652 _POSIX_VERSION == 200112L, plus (if the XSI option is supported) _XOPEN_UNIX and
16653 _XOPEN_VERSION == 600

16654 POSIX.1-2008, POSIX.1-2017, and Single UNIX Specification, Version 4
16655 _POSIX_VERSION == 200809L, plus (if the XSI option is supported) _XOPEN_UNIX and
16656 _XOPEN_VERSION == 700

16657 Note that _POSIX_VERSION did not change in POSIX.1-2017 as it was technically identical to
16658 POSIX.1-2008 with its two technical corrigenda applied.

16659 POSIX.1-2024 does not make any attempt to define application binary interaction with the
16660 underlying operating system. However, application developers may find it useful to query
16661 _SC_VERSION at runtime via *sysconf*( ) to determine whether the current version of the
16662 operating system supports the necessary functionality as in the following program fragment:

```
16663 if (sysconf(_SC_VERSION) < 202405L) {
16664     fprintf(stderr, "POSIX.1-2024 system required, terminating \n");
16665     exit(1);
16666 }
```

16667 Implementations may support multiple programming environments with some of them
16668 conforming to this standard and some not conforming. The _POSIX_V*n*_ILP* and
16669 _POSIX_V*n*_LP* constants, and corresponding *sysconf*( ) and *getconf* calls, only indicate whether

each programming environment is supported; they do not indicate anything about conformance of the environments that are supported. For example, an implementation may support the ILP32_OFF32 environment for legacy reasons with a 32-bit **time_t**, whereas in a conforming environment **time_t** is required to have a width of at least 64 bits. Application writers should consult an implementation's POSIX Conformance Document for information about the conformance of each supported programming environment.

New applications should not use _XOPEN_SHM or _XOPEN_ENH_I18N.

RATIONALE

As POSIX.1-2024 evolved, certain options became sufficiently standardized that it was concluded that simply requiring one of the option choices was simpler than retaining the option. However, for backwards-compatibility, the option flags (with required constant values) are retained.

Version Test Macros

The standard developers considered altering the definition of _POSIX_VERSION and removing _SC_VERSION from the specification of *sysconf*( ) since the utility to an application was deemed by some to be minimal, and since the implementation of the functionality is potentially problematic. However, they recognized that support for existing application binaries is a concern to manufacturers, application developers, and the users of implementations conforming to POSIX.1-2024.

While the example using _SC_VERSION in the APPLICATION USAGE section does not provide the greatest degree of imaginable utility to the application developer or user, it is arguably better than the production of a core image or some other equally obscure result. (It is also possible for implementations to encode and recognize application binaries compiled in various POSIX.1-conforming environments, and modify the semantics of the underlying system to conform to the expectations of the application.) For the reasons outlined in the preceding paragraphs and in the APPLICATION USAGE section, the standard developers elected to retain the _POSIX_VERSION and _SC_VERSION functionality.

Compile-Time Symbolic Constants for System-Wide Options

POSIX.1-2024 includes support in certain areas for the newly adopted policy governing options and stubs.

This policy provides flexibility for implementations in how they support options. It also specifies how conforming applications can adapt to different implementations that support different sets of options. It allows the following:

1. If an implementation has no interest in supporting an option, it does not have to provide anything associated with that option beyond the announcement that it does not support it.
2. An implementation can support a partial or incompatible version of an option (as a non-standard extension) as long as it does not claim to support the option.
3. An application can determine whether the option is supported. A strictly conforming application must check this announcement mechanism before first using anything associated with the option.

There is an important implication of this policy. POSIX.1-2024 cannot dictate the behavior of interfaces associated with an option when the implementation does not claim to support the option. In particular, it cannot require that a function associated with an unsupported option will fail if it does not perform as specified. However, this policy does not prevent a standard

from requiring certain functions to always be present, but that they shall always fail on some implementations. The *setpgid*() function in the POSIX.1-1990 standard, for example, is considered appropriate.

The POSIX standards include various options, and the C-language binding support for an option implies that the implementation must supply data types and function interfaces. An application must be able to discover whether the implementation supports each option.

Any application must consider the following three cases for each option:

1. Option never supported.

   The implementation advertises at compile time that the option will never be supported. In this case, it is not necessary for the implementation to supply any of the data types or function interfaces that are provided only as part of the option. The implementation might provide data types and functions that are similar to those defined by POSIX.1-2024, but there is no guarantee for any particular behavior.

2. Option always supported.

   The implementation advertises at compile time that the option will always be supported. In this case, all data types and function interfaces shall be available and shall operate as specified.

3. Option might or might not be supported.

   Some implementations might not provide a mechanism to specify support of options at compile time. In addition, the implementation might be unable or unwilling to specify support or non-support at compile time. In either case, any application that might use the option at runtime must be able to compile and execute. The implementation must provide, at compile time, all data types and function interfaces that are necessary to allow this. In this situation, there must be a mechanism that allows the application to query, at runtime, whether the option is supported. If the application attempts to use the option when it is not supported, the result is unspecified unless explicitly specified otherwise in POSIX.1-2024.

FUTURE DIRECTIONS

None.

SEE ALSO

<limits.h>, **<stddef.h>**, **<stdint.h>**, **<stdio.h>**, **<sys/socket.h>**, **<sys/types.h>**, **<termios.h>**, **<wctype.h>**

XSH *access*(), *alarm*(), *chown*(), *close*(), *confstr*(), *crypt*(), *ctermid*(), *dup*(), *_Exit*(), *encrypt*(), *exec*, *fchdir*(), *fchown*(), *fdatasync*(), *fork*(), *fpathconf*(), *fsync*(), *ftruncate*(), *getcwd*(), *getentropy*(), *getegid*(), *geteuid*(), *getgid*(), *getgroups*(), *gethostid*(), *gethostname*(), *getlogin*(), *getopt*(), *getpgid*(), *getpgrp*(), *getpid*(), *getppid*(), *getresgid*(), *getresuid*(), *getsid*(), *getuid*(), *isatty*(), *lchown*(), *link*(), *lockf*(), *lseek*(), *nice*(), *pause*(), *pipe*(), *read*(), *readlink*(), *rmdir*(), *setegid*(), *seteuid*(), *setgid*(), *setpgid*(), *setregid*(), *setresgid*(), *setresuid*(), *setreuid*(), *setsid*(), *setuid*(), *sleep*(), *swab*(), *symlink*(), *sync*(), *sysconf*(), *tcgetpgrp*(), *tcsetpgrp*(), *truncate*(), *ttyname*(), *unlink*(), *write*()

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

The symbolic constants _XOPEN_REALTIME and _XOPEN_REALTIME_THREADS are added.

_POSIX2_C_BIND, _XOPEN_ENH_I18N, and _XOPEN_SHM must now be set to a value other than –1 by a conforming implementation.

Large File System extensions are added.

The type of the argument to *sbrk*( ) is changed from **int** to **intptr_t**.

XBS constants are added to the list of constants for Options and Option Groups, to the list of constants for the *confstr*( ) function, and to the list of constants to the *sysconf*( ) function. These are all marked EX.

Issue 6

_POSIX2_C_VERSION is removed.

The Open Group Corrigendum U026/4 is applied, adding the prototype for *fdatasync*( ).

The Open Group Corrigendum U026/1 is applied, adding the symbols _SC_XOPEN_LEGACY, _SC_XOPEN_REALTIME, and _SC_XOPEN_REALTIME_THREADS.

The symbols _XOPEN_STREAMS and _SC_XOPEN_STREAMS are added to support the XSI STREAMS Option Group.

Text in the DESCRIPTION relating to conformance requirements is moved elsewhere in IEEE Std 1003.1-2001.

The LEGACY symbol _SC_PASS_MAX is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The _CS_POSIX_* and _CS_XBS5_* constants are added for the *confstr*( ) function.
- The _SC_XBS5_* constants are added for the *sysconf*( ) function.
- The symbolic constants F_ULOCK, F_LOCK, F_TLOCK, and F_TEST are added.
- The **uid_t**, **gid_t**, **off_t**, **pid_t**, and **useconds_t** types are mandated.

The *gethostname*( ) prototype is added for sockets.

A new section is added for System-Wide Options.

Function prototypes for *setegid*( ) and *seteuid*( ) are added.

Option symbolic constants are added for _POSIX_ADVISORY_INFO, _POSIX_CPUTIME, _POSIX_SPAWN, _POSIX_SPORADIC_SERVER, _POSIX_THREAD_CPUTIME, _POSIX_THREAD_SPORADIC_SERVER, and _POSIX_TIMEOUTS, and *pathconf*( ) variables are added for _PC_ALLOC_SIZE_MIN, _PC_REC_INCR_XFER_SIZE, _PC_REC_MAX_XFER_SIZE, _PC_REC_MIN_XFER_SIZE, and _PC_REC_XFER_ALIGN for alignment with IEEE Std 1003.1d-1999.

The following are added for alignment with IEEE Std 1003.1j-2000:

- Option symbolic constants _POSIX_BARRIERS, _POSIX_CLOCK_SELECTION, _POSIX_MONOTONIC_CLOCK, _POSIX_READER_WRITER_LOCKS, _POSIX_SPIN_LOCKS, and _POSIX_TYPED_MEMORY_OBJECTS
- *sysconf*( ) variables _SC_BARRIERS, _SC_CLOCK_SELECTION, _SC_MONOTONIC_CLOCK, _SC_READER_WRITER_LOCKS, _SC_SPIN_LOCKS, and _SC_TYPED_MEMORY_OBJECTS

The _SC_XBS5 macros associated with the ISO/IEC 9899:1990 standard are marked LEGACY, and new equivalent _SC_V6 macros associated with the ISO/IEC 9899:1999 standard are

introduced.

The *getwd*( ) function is marked LEGACY.

The **restrict** keyword is added to the prototypes for *readlink*( ) and *swab*( ).

Constants for options are now harmonized, so when supported they take the year of approval of IEEE Std 1003.1-2001 as the value.

The following are added for alignment with IEEE Std 1003.1q-2000:

- Optional symbolic constants _POSIX_TRACE, _POSIX_TRACE_EVENT_FILTER, _POSIX_TRACE_LOG, and _POSIX_TRACE_INHERIT
- The *sysconf*( ) symbolic constants _SC_TRACE, _SC_TRACE_EVENT_FILTER, _SC_TRACE_LOG, and _SC_TRACE_INHERIT

The *brk*( ) and *sbrk*( ) LEGACY functions are removed.

The Open Group Base Resolution bwg2001-006 is applied, which reworks the XSI versioning information.

The Open Group Base Resolution bwg2001-008 is applied, changing the *namelen* parameter for *gethostname*( ) from **socklen_t** to **size_t**.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/2 is applied, changing ``Thread Stack Address Size'' to ``Thread Stack Size Attribute''.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/20 is applied, adding the _POSIX_IPV6, _SC_V6, and _SC_RAW_SOCKETS symbols.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/21 is applied, correcting the description in ``Constants for Functions'' for the _CS_POSIX_V6_LP64_OFF64_CFLAGS, _CS_POSIX_V6_LP64_OFF64_LDFLAGS, and _CS_POSIX_V6_LP64_OFF64_LIBS symbols.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/22 is applied, removing the shading for the _PC* and _SC* constants, since these are mandatory on all implementations.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/23 is applied, adding the _PC_SYMLINK_MAX and _SC_SYMLOOP_MAX constants.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/24 is applied, correcting the shading and margin code for the *fsync*( ) function.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/25 is applied, adding the following text to the APPLICATION USAGE section: ``New applications should not use _XOPEN_SHM or _XOPEN_ENH_I18N.''.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/29 is applied, clarifying the requirements for when constants for Options and Option Groups can be defined or undefined.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/30 is applied, changing the _V6_ILP32_OFF32, _V6_ILP32_OFFBIG, _V6_LP64_OFF64, and _V6_LPBIG_OFFBIG symbols to _POSIX_V6_ILP32_OFF32, _POSIX_V6_ILP32_OFFBIG, _POSIX_V6_LP64_OFF64, and _POSIX_V6_LPBIG_OFFBIG, respectively. This is for consistency with the *sysconf*( ) and *c99* reference pages.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/31 is applied, adding that the format of names of programming environments can be obtained using the *getconf* **–v** option.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/32 is applied, deleting the _SC_FILE_LOCKING, _SC_2_C_VERSION, and _SC_XOPEN_XCU_VERSION constants.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/33 is applied, adding _SC_SS_REPL_MAX, _SC_TRACE_EVENT_NAME_MAX, _SC_TRACE_NAME_MAX, _SC_TRACE_SYS_MAX, and _SC_TRACE_USER_EVENT_MAX to the list of symbolic constants for *sysconf*( ).

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/34 is applied, updating the prototype for the *symlink*( ) function to match that in the System Interfaces volume of IEEE Std 1003.1-2001.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/35 is applied, adding _PC_2_SYMLINKS to the symbolic constants list for *pathconf*( ). This corresponds to the definition of POSIX2_SYMLINKS in the Shell and Utilities volume of IEEE Std 1003.1-2001.

Issue 7

Austin Group Interpretations 1003.1-2001 #026 and #047 are applied.

Austin Group Interpretation 1003.1-2001 #166 is applied to permit an additional compiler flag to enable threads.

Austin Group Interpretation 1003.1-2001 #178 is applied, clarifying the values allowed for _POSIX2_CHAR_TERM.

SD5-XBD-ERN-41 is applied, adding the _POSIX2_SYMLINKS constant.

SD5-XBD-ERN-76 and SD5-XBD-ERN-77 are applied.

Symbols to support the UUCP Utilities option are added.

The variables for the supported programming environments are updated to be V7.

The LEGACY and obsolescent symbols are removed.

The *faccessat*( ), *fchownat*( ), *fexecve*( ), *linkat*( ), *readlinkat*( ), *symlinkat*( ), and *unlinkat*( ) functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The _POSIX_TRACE* constants from the Trace option are marked obsolescent.

The _POSIX2_PBS* constants from the Batch Environment Services and Utilities option are marked obsolescent.

Functionality relating to the Asynchronous Input and Output, Barriers, Clock Selection, Memory Mapped Files, Memory Protection, Realtime Signals Extension, Semaphores, Spin Locks, Threads, Timeouts, and Timers options is moved to the Base.

Functionality relating to the Thread Priority Protection and Thread Priority Inheritance options is changed to be Non-Robust Mutex or Robust Mutex Priority Protection and Non-Robust Mutex or Robust Mutex Priority Inheritance, respectively.

This reference page is clarified with respect to macros and symbolic constants.

Changes are made related to support for finegrained timestamps and the _POSIX_TIMESTAMP_RESOLUTION constant is added.

The _SC_THREAD_ROBUST_PRIO_INHERIT and _SC_THREAD_ROBUST_PRIO_PROTECT symbolic constants are added.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0078 [311], XBD/TC1-2008/0079 [209], and XBD/TC1-2008/0080 [360] are applied.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0085 [783], XBD/TC2-2008/0086 [911], and XBD/TC2-2008/0087 [566] are applied.

Issue 8

Austin Group Defect 62 is applied, adding the *_Fork*( ) function.

Austin Group Defect 339 is applied, adding _SC_NPROCESSORS_CONF and _SC_NPROCESSORS_ONLN.

Austin Group Defects 411 and 598 are applied, adding *dup3*( ) and *pipe2*( ).

Austin Group Defects 415 and 1357 are applied, adding SEEK_HOLE and SEEK_DATA.

Austin Group Defect 529 is applied, adding the POSIX_CLOSE_RESTART symbolic constant and the *posix_close*( ) function.

Austin Group Defect 687 is applied, adding _POSIX_FALLOC and _PC_FALLOC.

Austin Group Defect 729 is applied, adding _POSIX_DEVICE_CONTROL and _SC_DEVICE_CONTROL.

Austin Group Defect 741 is applied, adding _SC_NSIG.

Austin Group Defects 1074 and 1116 are applied, changing the descriptions of _CS_POSIX_V8_THREADS_CFLAGS and _CS_POSIX_V8_THREADS_LDFLAGS.

Austin Group Defect 1122 is applied, adding _PC_TEXTDOMAIN_MAX.

Austin Group Defect 1134 is applied, adding *getentropy*( ).

Austin Group Defect 1141 is applied, changing the RATIONALE section.

Austin Group Defect 1192 is applied, marking the *encrypt*( ) function as obsolescent.

Austin Group Defect 1302 is applied, adding `_Noreturn` to *_exit*( ).

Austin Group Defects 1330 and 1595 are applied, removing obsolescent interfaces and changing ``_V7_'' to ``_V8_'' and ``_V6_'' to ``_V7_''.

Austin Group Defects 1344 and 1666 are applied, adding *getresgid*( ), *getresuid*( ), *setresgid*( ), and *setresuid*( ).

Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.

Austin Group Defect 1456 is applied, clarifying the reservation of symbolic constants with the prefix _CS_, _PC_, and _SC_.

Austin Group Defect 1462 is applied, adding a paragraph to APPLICATION USAGE about conformance of supported programming environments.

Austin Group Defect 1473 is applied, updating the list of earlier versions of this standard in the APPLICATION USAGE section.

Austin Group Defect 1518 is applied, correcting the spelling of ``programming''.

Austin Group Defect 1569 is applied, changing references to a **pointer** type to ``all pointer types''.

NAME

utmpx.h — user accounting database definitions

SYNOPSIS

XSI
```
#include <utmpx.h>
```

DESCRIPTION

The **<utmpx.h>** header shall define the **utmpx** structure that shall include at least the following members:

`char`	`ut_user[]`	User login name.
`char`	`ut_id[]`	Unspecified initialization process identifier.
`char`	`ut_line[]`	Device name.
`pid_t`	`ut_pid`	Process ID.
`short`	`ut_type`	Type of entry.
`struct timeval`	`ut_tv`	Time entry was made.

The **<utmpx.h>** header shall define the **pid_t** type through **typedef**, as described in **<sys/types.h>**.

The **<utmpx.h>** header shall define the **timeval** structure as described in **<sys/time.h>**.

Inclusion of the **<utmpx.h>** header may also make visible all symbols from **<sys/time.h>**.

The **<utmpx.h>** header shall define the following symbolic constants as possible values for the *ut_type* member of the **utmpx** structure:

EMPTY	No valid user accounting information.
BOOT_TIME	Identifies time of system boot.
OLD_TIME	Identifies time when system clock changed.
NEW_TIME	Identifies time after system clock changed.
USER_PROCESS	Identifies a process.
INIT_PROCESS	Identifies a process spawned by the init process.
LOGIN_PROCESS	Identifies the session leader of a logged-in user.
DEAD_PROCESS	Identifies a session leader who has exited.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
void          endutxent(void);
struct utmpx *getutxent(void);
struct utmpx *getutxid(const struct utmpx *);
struct utmpx *getutxline(const struct utmpx *);
struct utmpx *pututxline(const struct utmpx *);
void          setutxent(void);
```


16951 **APPLICATION USAGE**

16952 None.

16953 **RATIONALE**

16954 None.

16955 **FUTURE DIRECTIONS**

16956 None.

16957 **SEE ALSO**

16958 **<sys/time.h>**, **<sys/types.h>**

16959 XSH *endutxent*( )

16960 **CHANGE HISTORY**

16961 First released in Issue 4, Version 2.

NAME

wchar.h — wide-character handling

SYNOPSIS

```
#include <wchar.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to enable the visibility of these symbols in this header.

The **<wchar.h>** header shall define the following types:

CX **FILE** As described in **<stdio.h>**.

CX **locale_t** As described in **<locale.h>**.

mbstate_t A complete object type other than an array type that can hold the conversion state information necessary to convert between sequences of (possibly multi-byte) characters and wide characters. CX If a codeset is being used such that an **mbstate_t** needs to preserve more than two levels of reserved state, the results are unspecified.

size_t As described in **<stddef.h>**.

CX **va_list** As described in **<stdarg.h>**.

wchar_t As described in **<stddef.h>**.

wint_t An integer type capable of storing any valid value of **wchar_t** or WEOF.

The tag **tm** shall be declared as naming an incomplete structure type, the contents of which are described in the **<time.h>** header.

The implementation shall support one or more programming environments in which the width of **wint_t** is no greater than the width of type **long**. The names of these programming environments can be obtained using the *confstr*( ) function or the *getconf* utility.

The **<wchar.h>** header shall define the following macros:

WCHAR_MAX As described in **<stdint.h>**.

WCHAR_MIN As described in **<stdint.h>**.

WEOF Constant expression of type **wint_t** that is returned by several WP functions to indicate end-of-file.

NULL As described in **<stddef.h>**.

CX Inclusion of the **<wchar.h>** header may make visible all symbols from the headers **<ctype.h>**, **<string.h>**, **<stdarg.h>**, **<stddef.h>**, **<stdio.h>**, **<stdlib.h>**, and **<time.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers. Arguments to functions in this list can point to arrays containing **wchar_t** values that do not correspond to members of the character set of the current locale. Such values shall be processed according to the specified semantics, unless otherwise stated.

```
wint_t     btowc(int);
wint_t     fgetwc(FILE *);
wchar_t   *fgetws(wchar_t *restrict, int, FILE *restrict);
wint_t     fputwc(wchar_t, FILE *);
```



```
17004          int             fputws(const wchar_t *restrict, FILE *restrict);
17005          int             fwide(FILE *, int);
17006          int             fwprintf(FILE *restrict, const wchar_t *restrict, ...);
17007          int             fwscanf(FILE *restrict, const wchar_t *restrict, ...);
17008          wint_t          getwc(FILE *);
17009          wint_t          getwchar(void);
17010          size_t          mbrlen(const char *restrict, size_t, mbstate_t *restrict);
17011          size_t          mbrtowc(wchar_t *restrict, const char *restrict, size_t,
17012                              mbstate_t *restrict);
17013          int             mbsinit(const mbstate_t *);
17014   CX     size_t          mbsnrtowcs(wchar_t *restrict, const char **restrict,
17015                              size_t, size_t, mbstate_t *restrict);
17016          size_t          mbsrtowcs(wchar_t *restrict, const char **restrict, size_t,
17017                              mbstate_t *restrict);
17018   CX     FILE           *open_wmemstream(wchar_t **, size_t *);
17019          wint_t          putwc(wchar_t, FILE *);
17020          wint_t          putwchar(wchar_t);
17021          int             swprintf(wchar_t *restrict, size_t,
17022                              const wchar_t *restrict, ...);
17023          int             swscanf(const wchar_t *restrict,
17024                              const wchar_t *restrict, ...);
17025          wint_t          ungetwc(wint_t, FILE *);
17026          int             vfwprintf(FILE *restrict, const wchar_t *restrict, va_list);
17027          int             vfwscanf(FILE *restrict, const wchar_t *restrict, va_list);
17028          int             vswprintf(wchar_t *restrict, size_t,
17029                              const wchar_t *restrict, va_list);
17030          int             vswscanf(const wchar_t *restrict, const wchar_t *restrict,
17031                              va_list);
17032          int             vwprintf(const wchar_t *restrict, va_list);
17033          int             vwscanf(const wchar_t *restrict, va_list);
17034   CX     wchar_t        *wcpcpy(wchar_t *restrict, const wchar_t *restrict);
17035          wchar_t        *wcpncpy(wchar_t *restrict, const wchar_t *restrict, size_t);
17036          size_t          wcrtomb(char *restrict, wchar_t, mbstate_t *restrict);
17037   CX     int             wcscasecmp(const wchar_t *, const wchar_t *);
17038          int             wcscasecmp_l(const wchar_t *, const wchar_t *, locale_t);
17039          wchar_t        *wcscat(wchar_t *restrict, const wchar_t *restrict);
17040          wchar_t        *wcschr(const wchar_t *, wchar_t);
17041          int             wcscmp(const wchar_t *, const wchar_t *);
17042          int             wcscoll(const wchar_t *, const wchar_t *);
17043   CX     int             wcscoll_l(const wchar_t *, const wchar_t *, locale_t);
17044          wchar_t        *wcscpy(wchar_t *restrict, const wchar_t *restrict);
17045          size_t          wcscspn(const wchar_t *, const wchar_t *);
17046   CX     wchar_t        *wcsdup(const wchar_t *);
17047          size_t          wcsftime(wchar_t *restrict, size_t,
17048                              const wchar_t *restrict, const struct tm *restrict);
17049   CX     size_t          wcslcat(wchar_t *restrict, const wchar_t *restrict,
17050                              size_t);
17051          size_t          wcslcpy(wchar_t *restrict, const wchar_t *restrict,
17052                              size_t);
17053          size_t          wcslen(const wchar_t *);
17054   CX     int             wcsncasecmp(const wchar_t *, const wchar_t *, size_t);
17055          int             wcsncasecmp_l(const wchar_t *, const wchar_t *, size_t,
```



```
                    locale_t);
wchar_t      *wcsncat(wchar_t *restrict, const wchar_t *restrict, size_t);
int           wcsncmp(const wchar_t *, const wchar_t *, size_t);
wchar_t      *wcsncpy(wchar_t *restrict, const wchar_t *restrict, size_t);
CX  size_t    wcsnlen(const wchar_t *, size_t);
size_t        wcsnrtombs(char *restrict, const wchar_t **restrict, size_t,
                    size_t, mbstate_t *restrict);
wchar_t      *wcspbrk(const wchar_t *, const wchar_t *);
wchar_t      *wcsrchr(const wchar_t *, wchar_t);
size_t        wcsrtombs(char *restrict, const wchar_t **restrict,
                    size_t, mbstate_t *restrict);
size_t        wcsspn(const wchar_t *, const wchar_t *);
wchar_t      *wcsstr(const wchar_t *restrict, const wchar_t *restrict);
double        wcstod(const wchar_t *restrict, wchar_t **restrict);
float         wcstof(const wchar_t *restrict, wchar_t **restrict);
wchar_t      *wcstok(wchar_t *restrict, const wchar_t *restrict,
                    wchar_t **restrict);
long          wcstol(const wchar_t *restrict, wchar_t **restrict, int);
long double   wcstold(const wchar_t *restrict, wchar_t **restrict);
long long     wcstoll(const wchar_t *restrict, wchar_t **restrict, int);
unsigned long wcstoul(const wchar_t *restrict, wchar_t **restrict, int);
unsigned long long
              wcstoull(const wchar_t *restrict, wchar_t **restrict, int);
XSI int       wcswidth(const wchar_t *, size_t);
size_t        wcsxfrm(wchar_t *restrict, const wchar_t *restrict, size_t);
CX  size_t    wcsxfrm_l(wchar_t *restrict, const wchar_t *restrict,
                    size_t, locale_t);
int           wctob(wint_t);
XSI int       wcwidth(wchar_t);
wchar_t      *wmemchr(const wchar_t *, wchar_t, size_t);
int           wmemcmp(const wchar_t *, const wchar_t *, size_t);
wchar_t      *wmemcpy(wchar_t *restrict, const wchar_t *restrict, size_t);
wchar_t      *wmemmove(wchar_t *, const wchar_t *, size_t);
wchar_t      *wmemset(wchar_t *, wchar_t, size_t);
int           wprintf(const wchar_t *restrict, ...);
int           wscanf(const wchar_t *restrict, ...);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<ctype.h>, **<locale.h>**, **<stdarg.h>**, **<stddef.h>**, **<stdint.h>**, **<stdio.h>**, **<stdlib.h>**, **<string.h>**, **<time.h>**, **<wctype.h>**

XSH Section 2.2 (on page 496), *btowc( )*, *confstr( )*, *fgetwc( )*, *fgetws( )*, *fputwc( )*, *fputws( )*, *fwide( )*, *fwprintf( )*, *fwscanf( )*, *getwc( )*, *getwchar( )*, *iswalnum( )*, *iswalpha( )*, *iswcntrl( )*, *iswctype( )*, *iswdigit( )*, *iswgraph( )*, *iswlower( )*, *iswprint( )*, *iswpunct( )*, *iswspace( )*, *iswupper( )*, *iswxdigit( )*, *mbrlen( )*, *mbrtowc( )*, *mbsinit( )*, *mbsrtowcs( )*, *open_memstream( )*, *putwc( )*, *putwchar( )*, *towlower( )*,

17105 *towupper*( ), *ungetwc*( ), *vfwprintf*( ), *vfwscanf*( ), *wcrtomb*( ), *wcscasecmp*( ), *wcscat*( ), *wcschr*( ),
17106 *wcscmp*( ), *wcscoll*( ), *wcscpy*( ), *wcscspn*( ), *wcsdup*( ), *wcsftime*( ), *wcslcat*( ), *wcslen*( ), *wcsncat*( ),
17107 *wcsncmp*( ), *wcsncpy*( ), *wcspbrk*( ), *wcsrchr*( ), *wcsrtombs*( ), *wcsspn*( ), *wcsstr*( ), *wcstod*( ), *wcstok*( ),
17108 *wcstol*( ), *wcstoul*( ), *wcswidth*( ), *wcsxfrm*( ), *wctob*( ), *wctype*( ), *wcwidth*( ), *wmemchr*( ), *wmemcmp*( ),
17109 *wmemcpy*( ), *wmemmove*( ), *wmemset*( )

17110 XCU *getconf*

17111 **CHANGE HISTORY**

17112 First released in Issue 4.

17113 **Issue 5**

17114 Aligned with the ISO/IEC 9899: 1990/Amendment 1: 1995 (E).

17115 **Issue 6**

17116 The Open Group Corrigendum U021/10 is applied. The prototypes for *wcswidth*( ) and
17117 *wcwidth*( ) are marked as extensions.

17118 The Open Group Corrigendum U028/5 is applied, correcting the prototype for the *mbsinit*( )
17119 function.

17120 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

17121 • Various function prototypes are updated to add the **restrict** keyword.

17122 • The functions *vfwscanf*( ), *vswscanf*( ), *wcstof*( ), *wcstold*( ), *wcstoll*( ), and *wcstoull*( ) are
17123 added.

17124 The type **wctype_t**, the *isw**( ), *to**( ), and *wctype*( ) functions are marked as XSI extensions.

17125 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/26 is applied, adding the APPLICATION
17126 USAGE section.

17127 **Issue 7**

17128 The *mbsnrtowcs*( ), *open_wmemstream*( ), *wcpcpy*( ), *wcpncpy*( ), *wcscasecmp*( ), *wcsdup*( ),
17129 *wcsncasecmp*( ), *wcsnlen*( ), and *wcsnrtombs*( ) functions are added from The Open Group
17130 Technical Standard, 2006, Extended API Set Part 1.

17131 The *wcscasecmp_l*( ), *wcsncasecmp_l*( ), *wcscoll_l*( ), and *wcsxfrm_l*( ) functions are added from The
17132 Open Group Technical Standard, 2006, Extended API Set Part 4.

17133 The **wctype_t** type, and the *isw**( ), *towlower*( ), and *towupper*( ) functions are marked obsolescent
17134 in **<wchar.h>** since the ISO C standard requires the declarations to be in **<wctype.h>**.

17135 This reference page is clarified with respect to macros and symbolic constants, and a declaration
17136 for the **locale_t** type is added.

17137 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0081 [380] is applied.

17138 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0088 [73] is applied.

17139 **Issue 8**

17140 Austin Group Defect 986 is applied, adding *wcslcat*( ) and *wcslcpy*( ).

17141 Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899: 2018 standard.

17142 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

17143 Austin Group Defect 1352 is applied, changing the APPLICATION USAGE and RATIONALE
17144 sections.

NAME

wctype.h — wide-character classification and mapping utilities

SYNOPSIS

```
#include <wctype.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to enable the visibility of these symbols in this header.

The **<wctype.h>** header shall define the following types:

wint_t As described in **<wchar.h>**.

wctrans_t A scalar type that can hold values which represent locale-specific character mappings.

wctype_t A scalar type of a data object that can hold values which represent locale-specific character classification.

CX The **<wctype.h>** header shall define the **locale_t** type as described in **<locale.h>**.

The **<wctype.h>** header shall define the following macro:

WEOF As described in **<wchar.h>**.

For all functions described in this header that accept an argument of type **wint_t**, the value is representable as a **wchar_t** or equals the value of WEOF. If this argument has any other value, the behavior is undefined.

The behavior of these functions shall be affected by the *LC_CTYPE* category of the current locale.

CX Inclusion of the **<wctype.h>** header may make visible all symbols from the headers **<ctype.h>**, **<stdarg.h>**, **<stddef.h>**, **<stdio.h>**, **<stdlib.h>**, **<string.h>**, **<time.h>**, and **<wchar.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

```
int       iswalnum(wint_t);
CX int    iswalnum_l(wint_t, locale_t);
int       iswalpha(wint_t);
CX int    iswalpha_l(wint_t, locale_t);
int       iswblank(wint_t);
CX int    iswblank_l(wint_t, locale_t);
int       iswcntrl(wint_t);
CX int    iswcntrl_l(wint_t, locale_t);
int       iswctype(wint_t, wctype_t);
CX int    iswctype_l(wint_t, wctype_t, locale_t);
int       iswdigit(wint_t);
CX int    iswdigit_l(wint_t, locale_t);
int       iswgraph(wint_t);
CX int    iswgraph_l(wint_t, locale_t);
int       iswlower(wint_t);
CX int    iswlower_l(wint_t, locale_t);
int       iswprint(wint_t);
CX int    iswprint_l(wint_t, locale_t);
int       iswpunct(wint_t);
```



```
17189 CX      int        iswpunct_l(wint_t, locale_t);
17190         int        iswspace(wint_t);
17191 CX      int        iswspace_l(wint_t, locale_t);
17192         int        iswupper(wint_t);
17193 CX      int        iswupper_l(wint_t, locale_t);
17194         int        iswxdigit(wint_t);
17195 CX      int        iswxdigit_l(wint_t, locale_t);
17196         wint_t     towctrans(wint_t, wctrans_t);
17197 CX      wint_t     towctrans_l(wint_t, wctrans_t, locale_t);
17198         wint_t     towlower(wint_t);
17199 CX      wint_t     towlower_l(wint_t, locale_t);
17200         wint_t     towupper(wint_t);
17201 CX      wint_t     towupper_l(wint_t, locale_t);
17202         wctrans_t  wctrans(const char *);
17203 CX      wctrans_t  wctrans_l(const char *, locale_t);
17204         wctype_t   wctype(const char *);
17205 CX      wctype_t   wctype_l(const char *, locale_t);
```

17206 **APPLICATION USAGE**

17207 None.

17208 **RATIONALE**

17209 None.

17210 **FUTURE DIRECTIONS**

17211 None.

17212 **SEE ALSO**

17213 **<ctype.h>**, **<locale.h>**, **<stdarg.h>**, **<stddef.h>**, **<stdio.h>**, **<stdlib.h>**, **<string.h>**, **<time.h>**,
17214 **<wchar.h>**

17215 XSH Section 2.2 (on page 496), *iswalnum*( ), *iswalpha*( ), *iswblank*( ), *iswcntrl*( ), *iswctype*( ),
17216 *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ),
17217 *setlocale*( ), *towctrans*( ), *towlower*( ), *towupper*( ), *wctrans*( ), *wctype*( )

17218 **CHANGE HISTORY**

17219 First released in Issue 5. Derived from the ISO/IEC 9899: 1990/Amendment 1: 1995 (E).

17220 **Issue 6**

17221 The *iswblank*( ) function is added for alignment with the ISO/IEC 9899: 1999 standard.

17222 **Issue 7**

17223 SD5-XBD-ERN-6 is applied.

17224 The *_l( ) functions are added from The Open Group Technical Standard, 2006, Extended API Set
17225 Part 4.

17226 This reference page is clarified with respect to macros and symbolic constants.

17227 **Issue 8**

17228 Austin Group Defect 1330 is applied, moving the description of **wctype_t** from **<wchar.h>** to
17229 **<wctype.h>**.

17230 **NAME**

17231 wordexp.h — word-expansion types

17232 **SYNOPSIS**

17233 `#include <wordexp.h>`

17234 **DESCRIPTION**

17235 The **<wordexp.h>** header shall define the structures and symbolic constants used by the
17236 *wordexp*( ) and *wordfree*( ) functions.

17237 The **<wordexp.h>** header shall define the **wordexp_t** structure type, which shall include at least
17238 the following members:

17239 `size_t  we_wordc` Count of words matched by *words.*
17240 `char  **we_wordv` Pointer to list of expanded words.
17241 `size_t  we_offs` Slots to reserve at the beginning of *we_wordv.*

17242 The **<wordexp.h>** header shall define the following symbolic constants for use as flags for the
17243 *wordexp*( ) function:

17244 WRDE_APPEND Append words to those previously generated.

17245 WRDE_DOOFFS Number of null pointers to prepend to *we_wordv*.

17246 WRDE_NOCMD Fail if command substitution is requested.

17247 WRDE_REUSE The *pwordexp* argument was passed to a previous successful call to
17248 *wordexp*( ), and has not been passed to *wordfree*( ). The result is the same
17249 as if the application had called *wordfree*( ) and then called *wordexp*( )
17250 without WRDE_REUSE.

17251 WRDE_SHOWERR Do not redirect *stderr* to **/dev/null**.

17252 WRDE_UNDEF Report error on an attempt to expand an undefined shell variable.

17253 The **<wordexp.h>** header shall define the following symbolic constants as error return values:

17254 WRDE_BADCHAR One of the unquoted characters—<newline>, `'|'`, `'&'`, `';'`, `'<'`, `'>'`,
17255 `'('`,`')'`,`'{'`,`'}'`—appears in *words* in an inappropriate context.

17256 WRDE_BADVAL Reference to undefined shell variable when WRDE_UNDEF is set in *flags*.

17257 WRDE_CMDSUB Command substitution requested when WRDE_NOCMD was set in *flags*.

17258 WRDE_NOSPACE Attempt to allocate memory failed.

17259 WRDE_SYNTAX Shell syntax error, such as unbalanced parentheses or unterminated
17260 string.

17261 The **<wordexp.h>** header shall define the **size_t** type as described in **<stddef.h>**.

17262 The following shall be declared as functions and may also be defined as macros. Function
17263 prototypes shall be provided.

```
17264 int  wordexp(const char *restrict, wordexp_t *restrict, int);
17265 void wordfree(wordexp_t *);
```


APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<stddef.h>

XSH *wordexp*( )

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 6

The **restrict** keyword is added to the prototype for *wordexp*( ).

The WRDE_NOSYS constant is marked obsolescent.

Issue 7

The obsolescent WRDE_NOSYS constant is removed.

This reference page is clarified with respect to macros and symbolic constants.

Issue 8

Austin Group Defect 1444 is applied, correcting cross-references to *wordexp*( ).

17285 *The Open Group Standard*

17286 **Vol. 2:**

17287 **System Interfaces, Issue 8**

17288 *The Open Group*
17289 *The Institute of Electrical and Electronics Engineers, Inc.*

17290 *Chapter 1*

17291 # *Introduction*

17292 The System Interfaces volume of POSIX.1-2024 describes the interfaces offered to application
17293 programs by POSIX-conformant systems.

17294 ## 1.1 Relationship to Other Formal Standards

17295 This volume of POSIX.1-2024 is aligned with the following standards, except where stated
17296 otherwise:

17297 ISO C (C17)
17298 ISO/IEC 9899: 2018, Programming Languages — C.

17299 Parts of the ISO/IEC 9899: 2018 standard (hereinafter referred to as the ISO C standard) are
17300 referenced to describe requirements also mandated by this volume of POSIX.1-2024. Some
17301 functions and headers included within this volume of POSIX.1-2024 have a version in the ISO C
17302 standard; in this case CX markings are added as appropriate to show where the ISO C standard
17303 has been extended (see Section 1.8.1, on page 7). Any conflict between this volume of
17304 POSIX.1-2024 and the ISO C standard is unintentional, except where stated otherwise.

17305 This volume of POSIX.1-2024 also allows, but does not require, mathematics functions to
17306 support IEEE Std 754-1985 and IEEE Std 854-1987.

17307 ## 1.2 Format of Entries

17308 The entries in Chapter 3 are based on a common format as follows. The only sections relating to
17309 conformance are the SYNOPSIS, DESCRIPTION, RETURN VALUE, and ERRORS sections.

17310 **NAME**
17311 This section gives the name or names of the entry and briefly states its purpose.

17312 **SYNOPSIS**
17313 This section summarizes the use of the entry being described. If it is necessary to
17314 include a header to use this function, the names of such headers are shown, for
17315 example:

17316 ```
#include <stdio.h>
```

17317 **DESCRIPTION**
17318 This section describes the functionality of the function or header.

17319 **RETURN VALUE**
17320 This section indicates the possible return values, if any.

17321 If the implementation can detect errors, ``successful completion'' means that no error
17322 has been detected during execution of the function. If the implementation does detect
17323 an error, the error is indicated.

17324 For functions where no errors are defined, ``successful completion'' means that if the

  
```

17325 implementation checks for errors, no error has been detected. If the implementation can
17326 detect errors, and an error is detected, the indicated return value is returned and *errno*
17327 may be set.

17328 **ERRORS**

17329 This section gives the symbolic names of the error values returned by a function or
17330 stored into a variable accessed through the symbol *errno* if an error occurs.

17331 ``No errors are defined'' means that error values returned by a function or stored into a
17332 variable accessed through the symbol *errno*, if any, depend on the implementation.

17333 **EXAMPLES**

17334 This section is informative.

17335 This section gives examples of usage, where appropriate. In the event of conflict
17336 between an example and a normative part of this volume of POSIX.1-2024, the
17337 normative material is to be taken as correct.

17338 **APPLICATION USAGE**

17339 This section is informative.

17340 This section gives warnings and advice to application developers about the entry. In the
17341 event of conflict between warnings and advice and a normative part of this volume of
17342 POSIX.1-2024, the normative material is to be taken as correct.

17343 **RATIONALE**

17344 This section is informative.

17345 This section contains historical information concerning the contents of this volume of
17346 POSIX.1-2024 and why features were included or discarded by the standard
17347 developers.

17348 **FUTURE DIRECTIONS**

17349 This section is informative.

17350 This section provides comments which should be used as a guide to current thinking;
17351 there is not necessarily a commitment to adopt these future directions.

17352 **SEE ALSO**

17353 This section is informative.

17354 This section gives references to related information.

17355 **CHANGE HISTORY**

17356 This section is informative.

17357 This section shows the derivation of the entry and any significant changes that have
17358 been made to it.

17359 Chapter 2

17360 # General Information

17361 This chapter covers information that is relevant to all the functions specified in Chapter 3 and
17362 XBD Chapter 14 (on page 221).

17363 ## 2.1 Use and Implementation of Interfaces

17364 ### 2.1.1 Use and Implementation of Functions

17365 Each of the following statements shall apply to all functions unless explicitly stated otherwise in
17366 the detailed descriptions that follow:

17367 1. If an argument to a function has an invalid value, such as a value outside the domain of
17368 the function, a pointer to an object whose lifetime has ended (even if a new object now
17369 has the same address), a pointer outside the address space of the program, or a null
17370 pointer, the behavior is undefined.

17371 2. Any function declared in a header may also be implemented as a macro defined in the
17372 header, so a function should not be declared explicitly if its header is included. Any
17373 macro definition of a function can be suppressed locally by enclosing the name of the
17374 function in parentheses, because the name is then not followed by the <left-parenthesis>
17375 that indicates expansion of a macro function name. For the same syntactic reason, it is
17376 permitted to take the address of a function even if it is also defined as a macro. The use of
17377 the C-language **#undef** construct to remove any such macro definition shall also ensure
17378 that an actual function is referred to.

17379 3. Any invocation of a function that is implemented as a macro shall expand to code that
17380 evaluates each of its arguments exactly once, fully protected by parentheses where
17381 necessary, so it is generally safe to use arbitrary expressions as arguments.

17382 4. For functions from the ISO C standard only, provided that the function can be declared
17383 without reference to any type defined in a header from the ISO C standard, it is also
17384 permissible to declare the function explicitly and use it without including its associated
17385 header.

17386 5. If a function that accepts a variable number of arguments is not declared (explicitly or by
17387 including its associated header), the behavior is undefined.

17388 6. Functions shall prevent data races as follows: A function shall not directly or indirectly
17389 access objects accessible by threads other than the current thread unless the objects are
17390 accessed directly or indirectly via the function's arguments. A function shall not directly
17391 or indirectly modify objects accessible by threads other than the current thread unless the
17392 objects are accessed directly or indirectly via the function's non-const arguments.
17393 Implementations may share their own internal objects between threads if the objects are
17394 not visible to applications and are protected against data races.

7. Functions shall perform all operations solely within the current thread if those operations have effects that are visible to applications.

2.1.2 Use and Implementation of Macros

Each of the following statements shall apply to all macros unless explicitly stated otherwise:

1. Any definition of an object-like macro in a header shall expand to code that is fully protected by parentheses where necessary, so that it groups in an arbitrary expression as if it were a single identifier.
2. All object-like macros listed as expanding to integer constant expressions shall additionally be suitable for use in **#if** preprocessing directives.
3. Any definition of a function-like macro in a header shall expand to code that evaluates each of its arguments exactly once, fully protected by parentheses where necessary, so that it is generally safe to use arbitrary expressions as arguments.
4. Any definition of a function-like macro in a header can be invoked in an expression anywhere a function with a compatible return type could be called.

2.2 The Compilation Environment

2.2.1 POSIX.1 Symbols

Certain symbols in this volume of POSIX.1-2024 are defined in headers (see XBD Chapter 14, on page 221). Some of those headers could also define symbols other than those defined by POSIX.1-2024, potentially conflicting with symbols used by the application. Also, POSIX.1-2024 defines symbols that are not permitted by other standards to appear in those headers without some control on the visibility of those symbols.

Symbols called ``feature test macros'' are used to control the visibility of symbols that might be included in a header. Implementations, future versions of this standard, and other standards may define additional feature test macros.

In the compilation of an application that **#define**s a feature test macro specified by POSIX.1-2024, no header defined by POSIX.1-2024 shall be included prior to the definition of the feature test macro. This restriction also applies to any implementation-provided header in which these feature test macros are used. If the definition of the macro does not precede the **#include**, the result is undefined.

Feature test macros shall begin with the <underscore> character (`'_'`).

2.2.1.1 *The _POSIX_C_SOURCE Feature Test Macro*

A POSIX-conforming application shall ensure that the feature test macro _POSIX_C_SOURCE is defined before inclusion of any header.

When an application includes a header described by POSIX.1-2024, and when this feature test macro is defined to have the value 202405L:

17430 1. All symbols required by POSIX.1-2024 to appear when the header is included shall be
17431 made visible.

17432 2. Symbols that are explicitly permitted, but not required, by POSIX.1-2024 to appear in that
17433 header (including those in reserved name spaces) may be made visible.

17434 3. Additional symbols not required or explicitly permitted by POSIX.1-2024 to be in that
17435 header shall not be made visible, except when enabled by another feature test macro.

17436 Identifiers in POSIX.1-2024 may only be undefined using the **#undef** directive as described in
17437 Section 2.1 (on page 495) or Section 2.2.2 (on page 498). These **#undef** directives shall follow all
17438 **#include** directives of any header in POSIX.1-2024.

17439 **Note:** The POSIX.1-1990 standard specified a macro called _POSIX_SOURCE. This has been
17440 superseded by _POSIX_C_SOURCE.

17441 *2.2.1.2 The _XOPEN_SOURCE Feature Test Macro*

17442 XSI An XSI-conforming application shall ensure that the feature test macro _XOPEN_SOURCE is
17443 defined with the value 800 before inclusion of any header. This is needed to enable the
17444 functionality described in Section 2.2.1.1 (on page 496) and to ensure that the XSI option is
17445 enabled.

17446 Since this volume of POSIX.1-2024 is aligned with the ISO C standard, and since all functionality
17447 enabled by _POSIX_C_SOURCE set equal to 202405L is enabled by _XOPEN_SOURCE set equal
17448 to 800, there should be no need to define _POSIX_C_SOURCE if _XOPEN_SOURCE is so
17449 defined. Therefore, if _XOPEN_SOURCE is set equal to 800 and _POSIX_C_SOURCE is set equal
17450 to 202405L, the behavior is the same as if only _XOPEN_SOURCE is defined and set equal to
17451 800. However, should _POSIX_C_SOURCE be set to a value greater than 202405L, the behavior
17452 is unspecified.

17453 If _XOPEN_SOURCE is defined with the value 800 and _POSIX_C_SOURCE is undefined before
17454 inclusion of any header, then the header may define the _POSIX_C_SOURCE macro with the
17455 value 202405L.

17456 *2.2.1.3 The __STDC_WANT_LIB_EXT1__ Feature Test Macro*

17457 XSI A POSIX-conforming or XSI-conforming application can define the feature test macro
17458 __STDC_WANT_LIB_EXT1__ before inclusion of any header.

17459 When an application includes a header described by POSIX.1-2024, and when this feature test
17460 macro is defined to have the value 1, the header may make visible those symbols specified for
17461 the header in Annex K of the ISO C standard that are not already explicitly permitted by
17462 POSIX.1-2024 to be made visible in the header. These symbols are listed in Section 2.2.2 below.

17463 When an application includes a header described by POSIX.1-2024, and when this feature test
17464 macro is either undefined or defined to have the value 0, the header shall not make any
17465 additional symbols visible that are not already made visible by the feature test macro
17466 XSI _POSIX_C_SOURCE or _XOPEN_SOURCE as described above, except when enabled by
17467 another feature test macro.

2.2.2 The Name Space

All identifiers in this volume of POSIX.1-2024, except *environ*, are defined in at least one of the
XSI headers, as shown in XBD Chapter 14 (on page 221). When _XOPEN_SOURCE or _POSIX_C_SOURCE is defined, each header defines or declares some identifiers, potentially conflicting with identifiers used by the application. The set of identifiers visible to the application consists of precisely those identifiers from the header pages of the included headers, as well as additional identifiers reserved for the implementation. In addition, some headers may make visible identifiers from other headers as indicated on the relevant header pages.

Implementations may also add members to a structure or union without controlling the visibility of those members with a feature test macro, as long as a user-defined macro with the same name cannot interfere with the correct interpretation of the program. The identifiers reserved for use by the implementation are described below:

1. Each identifier with external linkage described in the header section is reserved for use as an identifier with external linkage if the header is included.
2. Each macro described in the header section is reserved for any use if the header is included.
3. Each identifier with file scope described in the header section is reserved for use as a macro name and as an identifier with file scope in the same name space if the header is included.

As described in Chapter 13 (on page 219), the prefixes posix_, POSIX_, and _POSIX_ are reserved for use by POSIX.1-2024 and other POSIX standards. Implementations may add symbols to the headers shown in the following table, provided the identifiers for those symbols either:

1. Begin with the corresponding reserved prefixes in the table, or
2. Have one of the corresponding complete names in the table, or
3. End in the string indicated as a reserved suffix in the table and do not use the reserved prefixes posix_, POSIX_, or _POSIX_, as long as the reserved suffix is in that part of the name considered significant by the implementation.

Symbols that use the reserved prefix _POSIX_ may be made visible by implementations in any header defined by POSIX.1-2024.

	Header	Prefix	Suffix	Complete Name
	<aio.h>	aio_, lio_, AIO_, LIO_		
	<arpa/inet.h>	inet_		
	<ctype.h>	to[a-z], is[a-z]		
	<dlfcn.h>	RTLD_, dli_		
	<dirent.h>	d_, DT_		
	<fcntl.h>	l_		
XSI	**<fmtmsg.h>**	MM_		
	<fnmatch.h>	FNM_		
XSI	**<ftw.h>**	FTW		
	<glob.h>	gl_, GLOB_		
	<grp.h>	gr_		
	<libintl.h>			TEXTDOMAINMAX
	<limits.h>		_MAX, _MIN	
XSI	**<math.h>**	M_		
MSG	**<mqueue.h>**	mq_, MQ_		
XSI	**<ndbm.h>**	dbm_, DBM_		
	<netdb.h>	ai_, h_, n_, p_, s_		
	<net/if.h>	if_, IF_		
	<netinet/in.h>	in_, ip_, s_, sin_, INADDR_, IPPROTO_		
IP6		in6_, in6addr_, s6_, sin6_, IPV6_		
	<netinet/tcp.h>	TCP_		
	<nl_types.h>	NL_		
	<poll.h>	pd_, ph_, ps_, POLL		
	<pthread.h>	pthread_, PTHREAD_		
	<pwd.h>	pw_		
	<regex.h>	re_, rm_, REG_		
	<sched.h>	sched_, SCHED_		
	<semaphore.h>	sem_, SEM_		
CX	**<signal.h>**	sa_, si_, sigev_, sival_, uc_, BUS_, CLD_, FPE_, ILL_, SA_, SEGV_, SI_, SIGEV_, ss_, sv_, SS_, TRAP_		
	<stdatomic.h>	atomic_[a-z], memory_[a-z]		
	<stdlib.h>	str[a-z]		
	<string.h>	str[a-z], mem[a-z], wcs[a-z]		
XSI	**<sys/ipc.h>**	ipc_, IPC_		key, pad, seq
	<sys/mman.h>	shm_, MAP_, MCL_, MS_, PROT_		
XSI	**<sys/msg.h>**	msg, MSG_[A-Z]		msg
XSI	**<sys/resource.h>**	rlim_, ru_, PRIO_, RLIMIT_, RUSAGE_		
	<sys/select.h>	fd_, fds_, FD_		

	Header	Prefix	Suffix	Complete Name
XSI	**<sys/sem.h>**	sem, SEM_		sem
XSI	**<sys/shm.h>**	shm, SHM[A-Z], SHM_[A-Z]		
	<sys/socket.h>	cmsg_, if_, ifc_, ifra_, ifru_, infu_, l_, msg_, sa_, ss_,		
XSI		AF_, MSG_, PF_, SCM_, SHUT_, SO		
	<sys/stat.h>	st_		
	<sys/statvfs.h>	f_, ST_		
XSI	**<sys/time.h>**	tv_		
	<sys/times.h>	tms_		
XSI	**<sys/uio.h>**	iov_		UIO_MAXIOV
	<sys/un.h>	sun_		
	<sys/utsname.h>	uts_		
	<sys/wait.h>	P_, W[A-Z]		
XSI	**<syslog.h>**	LOG_		
	<termios.h>	c_, B[0-9], TC, ws_		
	<threads.h>	cnd_[a-z], mtx_[a-z], thrd_[a-z], tss_[a-z]		
CX	**<time.h>**	clock_, it_, timer_, tm_, tv_, CLOCK_, TIMER_		
XSI	**<utmpx.h>**	ut_	_LVL, _PROCESS, _TIME	
	<wchar.h>	wcs[a-z]		
	<wctype.h>	is[a-z], to[a-z]		
	<wordexp.h>	we_, WRDE_		
CX	**ANY header**		_t	

Note: The notation [0–9] indicates any digit. The notation [A–Z] indicates any uppercase letter in the portable character set. The notation [a–z] indicates any lowercase letter in the portable character set. Commas and spaces in the lists of prefixes and complete names in the above table are not part of any prefix or complete name. The ISO C standard reserves int[0-9a-z_]*_t and uint[0-9a-z_]*_t in **<stdint.h>**; this is not included in the table above because it is covered by the reserved _t suffix for any header.

Additional symbolic constants with the prefix _CS_, _PC_, and _SC_ may be defined by the inclusion of **<unistd.h>**, but as these are already reserved for the implementation, they are not included in the table above. Extensions with these prefixes should be compatible with use by *confstr*( ), *pathconf*( ), and *sysconf*( ), respectively.

Implementations may also add symbols to the **<complex.h>** header with the following complete names or the same names suffixed with `'f'` or `'l'`:

cerf	cerfc	cexp2
cexpm1	clog10	clog1p
clog2	clgamma	ctgamma

If any header in the following table is included, macros with the prefixes or suffixes shown may be defined. After the last inclusion of a given header, an application may use identifiers with the corresponding prefixes for its own purpose, provided their use is preceded by a **#undef** of the corresponding macro.

Header	Prefix	Suffix
<endian.h>		_ENDIAN
<errno.h>	E[0-9], E[A-Z]	
<fcntl.h>	F_, O_	
<fenv.h>	FE_[A-Z]	
<inttypes.h>	PRI[Xa-z], SCN[Xa-z]	
<locale.h>	LC_[A-Z]	
<math.h>	FP_[A-Z]	
<netinet/in.h>	IMPLINK_, IN_, IP_, IPPORT_, SOCK_,	
IP6	IN6_	
<signal.h>	SIG_, SIG[A-Z],	
XSI	SV_	
<stdatomic.h>	ATOMIC_[A-Z]	
CX **<stdio.h>**	SEEK_	
XSI **<sys/resource.h>**	RLIM_	
XSI **<sys/socket.h>**	CMSG_	
<sys/stat.h>	S_	
XSI **<sys/uio.h>**	IOV_	
<termios.h>	I, O, V (See below.)	
<time.h>	TIME_[A-Z]	
<unistd.h>	SEEK_	

The following are used to reserve complete names for the **<stdint.h>** header:

INT[0-9A-Za-z_]*_MIN
INT[0-9A-Za-z_]*_MAX
INT[0-9A-Za-z_]*_C
UINT[0-9A-Za-z_]*_MIN
UINT[0-9A-Za-z_]*_MAX
UINT[0-9A-Za-z_]*_C

Note: The notation [0–9] indicates any digit. The notation [A–Z] indicates any uppercase letter in the portable character set. The notation [Xa–z] indicates the character `'X'` or any lowercase letter in the portable character set. The notation [0–9A–Za–z_]* indicates zero or more occurrences of any of the following: a digit, an uppercase or lowercase letter in the portable character set, or an <underscore>.

XSI The following reserved names are used as exact matches for **<termios.h>**:

CBAUD	EXTB	VDSUSP
DEFECHO	FLUSHO	VLNEXT
ECHOCTL	LOBLK	VREPRINT
ECHOKE	PENDIN	VSTATUS
ECHOPRT	SWTCH	VWERASE
EXTA	VDISCARD	

When the feature test macro__STDC_WANT_LIB_EXT1__ is defined with the value 1 (see Section 2.2.1, on page 496), implementations may add symbols to the headers shown in the following table provided the identifiers for those symbols have one of the corresponding

17633 complete names in the table.

17634

Header	Complete Name
<stdio.h>	fopen_s, fprintf_s, freopen_s, fscanf_s, gets_s, printf_s, scanf_s, snprintf_s, sprintf_s, sscanf_s, tmpfile_s, tmpnam_s, vfprintf_s, vfscanf_s, vprintf_s, vscanf_s, vsnprintf_s, vsprintf_s, vsscanf_s
<stdlib.h>	abort_handler_s, bsearch_s, getenv_s, ignore_handler_s, mbstowcs_s, qsort_s, set_constraint_handler_s, wcstombs_s, wctomb_s
<time.h>	asctime_s, ctime_s, gmtime_s, localtime_s
<wchar.h>	fwprintf_s, fwscanf_s, mbsrtowcs_s, snwprintf_s, swprintf_s, swscanf_s, vfwprintf_s, vfwscanf_s, vsnwprintf_s, vswprintf_s, vswscanf_s, vwprintf_s, vwscanf_s, wcrtomb_s, wmemcpy_s, wmemmove_s, wprintf_s, wscanf_s

17646 When the feature test macro__STDC_WANT_LIB_EXT1__ is defined with the value 1 (see
17647 Section 2.2.1, on page 496), if any header in the following table is included, macros with the
17648 complete names shown may be defined.

17649

Header	Complete Name
<stdint.h>	RSIZE_MAX
<stdio.h>	L_tmpnam_s, TMP_MAX_S

17652 **Note:** The above two tables only include those symbols from Annex K of the ISO C standard that are
17653 not already allowed to be visible by entries in earlier tables in this section.

17654 The following identifiers are reserved regardless of the inclusion of headers:

17655 1. With the exception of identifiers beginning with the prefix _POSIX_ and those identifiers
17656 which are lexically identical to keywords defined by the ISO C standard (for example
17657 **_Bool**), all identifiers that begin with an <underscore> and either an uppercase letter or
17658 another <underscore> are always reserved for any use by the implementation.

17659 2. All identifiers that begin with an <underscore> are always reserved for use as identifiers
17660 with file scope in both the ordinary identifier and tag name spaces.

17661 3. All identifiers in the table below are reserved for use as identifiers with external linkage.
17662 Some of these identifiers do not appear in this volume of POSIX.1-2024, but are reserved for
17663 future use by the ISO C standard.

17664 4. All functions and external identifiers defined in XBD Chapter 14 (on page 221) are reserved
17665 for use as identifiers with external linkage.

17666 5. All the identifiers defined in this volume of POSIX.1-2024 that have external linkage and
17667 *errno* are always reserved for use as identifiers with external linkage.

17668 **Note:** The notation [a–z] indicates any lowercase letter in the portable character set. The notation `'*'`
17669 indicates any combination of digits, letters in the portable character set, or <underscore>.

17670 No other identifiers are reserved.

17671 _Exit
17672 abort
17673 abs
17674 acos
17675 acosf
17676 acosh
17677 acoshf
17678 acoshl
17679 acosl
17680 aligned_alloc
17681 asctime
17682 asin
17683 asinf
17684 asinh
17685 asinhf
17686 asinhl
17687 asinl
17688 at_quick_exit
17689 atan
17690 atan2
17691 atan2f
17692 atan2l
17693 atanf
17694 atanh
17695 atanhf
17696 atanhl
17697 atanl
17698 atexit
17699 atof
17700 atoi
17701 atol
17702 atoll
17703 atomic_compare_exchange_strong
17704 atomic_compare_exchange_strong_explicit
17705 atomic_compare_exchange_weak
17706 atomic_compare_exchange_weak_explicit
17707 atomic_exchange
17708 atomic_exchange_explicit
17709 atomic_fetch_add
17710 atomic_fetch_add_explicit
17711 atomic_fetch_and
17712 atomic_fetch_and_explicit
17713 atomic_fetch_or
17714 atomic_fetch_or_explicit
17715 atomic_fetch_sub
17716 atomic_fetch_sub_explicit
17717 atomic_fetch_xor
17718 atomic_fetch_xor_explicit
17719 atomic_flag_clear
17720 atomic_flag_clear_explicit
17721 atomic_flag_test_and_set
17722 atomic_flag_test_and_set_explicit
atomic_init
atomic_is_lock_free
atomic_load
atomic_load_explicit
atomic_signal_fence
atomic_store
atomic_store_explicit
atomic_thread_fence
bsearch
btowc
c16rtomb
c32rtomb
cabs
cabsf
cabsl
cacos
cacosf
cacosh
cacoshf
cacoshl
cacosl
call_once
calloc
carg
cargf
cargl
casin
casinf
casinh
casinhf
casinhl
casinl
catan
catanf
catanh
catanhf
catanhl
catanl
cbrt
cbrtf
cbrtl
ccos
ccosf
ccosh
ccoshf
ccoshl
ccosl
ceil
ceilf
ceill
cerf
cerfc
cerfcf
cerfcl
cerff
cerfl
cexpm1
cexpm1f
cexpm1l
cexp
cexp2
cexp2f
cexp2l
cexpf
cexpl
cimag
cimagf
cimagl
clearerr
clgamma
clgammaf
clgammal
clock
clog
clog10
clog10f
clog10l
clog1p
clog1pf
clog1pl
clog2
clog2f
clog2l
clogf
clogl
cnd_broadcast
cnd_destroy
cnd_init
cnd_signal
cnd_timedwait
cnd_wait
conj
conjf
conjl
copysign
copysignf
copysignl
cos
cosf
cosh
coshf
coshl
cosl
cpow

17723 cpowf
17724 cpowl
17725 cproj
17726 cprojf
17727 cprojl
17728 creal
17729 crealf
17730 creall
17731 csin
17732 csinf
17733 csinh
17734 csinhf
17735 csinhl
17736 csinl
17737 csqrt
17738 csqrtf
17739 csqrtl
17740 ctan
17741 ctanf
17742 ctanh
17743 ctanhf
17744 ctanhl
17745 ctanl
17746 ctgamma
17747 ctgammaf
17748 ctgammal
17749 ctime
17750 difftime
17751 div
17752 erf
17753 erfc
17754 erfcf
17755 erfcl
17756 erff
17757 erfl
17758 errno
17759 exit
17760 exp
17761 exp2
17762 exp2f
17763 exp2l
17764 expf
17765 expl
17766 expm1
17767 expm1f
17768 expm1l
17769 fabs
17770 fabsf
17771 fabsl
17772 fclose
17773 fdim
17774 fdimf
fdiml
feclearexcept
fegetenv
fegetexceptflag
fegetround
feholdexcept
feof
feraiseexcept
ferror
fesetenv
fesetexceptflag
fesetround
fetestexcept
feupdateenv
fflush
fgetc
fgetpos
fgets
fgetwc
fgetws
floor
floorf
floorl
fma
fmaf
fmal
fmax
fmaxf
fmaxl
fmin
fminf
fminl
fmod
fmodf
fmodl
fopen
fprintf
fputc
fputs
fputwc
fputws
fread
free
freopen
frexp
frexpf
frexpl
fscanf
fseek
fsetpos
ftell
fwide
fwprintf
fwrite
fwscanf
getc
getchar
getenv
getwc
getwchar
gmtime
hypot
hypotf
hypotl
ilogb
ilogbf
ilogbl
imaxabs
imaxdiv
is[a-z]*
kill_dependency
labs
ldexp
ldexpf
ldexpl
ldiv
lgamma
lgammaf
lgammal
llabs
lldiv
llrint
llrintf
llrintl
llround
llroundf
llroundl
localeconv
localtime
log
log10
log10f
log10l
log1p
log1pf
log1pl
log2
log2f
log2l
logb
logbf
logbl
logf
logl
longjmp
lrint
lrintf
lrintl
lround
lroundf
lroundl
malloc
math_errhandling
mblen
mbrlen
mbrtoc16
mbrtoc32
mbrtowc
mbsinit
mbsrtowcs
mbstowcs
mbtowc
mem[a-z]*
mktime
modf
modff
modfl
mtx_destroy
mtx_init
mtx_lock
mtx_timedlock
mtx_trylock
mtx_unlock
nan
nanf
nanl
nearbyint
nearbyintf
nearbyintl
nextafter
nextafterf
nextafterl
nexttoward
nexttowardf
nexttowardl
perror
pow
powf
powl
printf
putc
putchar
puts
putwc
putwchar
qsort

quick_exit	setjmp	tgamma	vfprintf
raise	setlocale	tgammaf	vfscanf
rand	setvbuf	tgammal	vfwprintf
realloc	signal	thrd_create	vfwscanf
remainder	sin	thrd_current	vprintf
remainderf	sinf	thrd_detach	vscanf
remainderl	sinh	thrd_equal	vsnprintf
remove	sinhf	thrd_exit	vsprintf
remquo	sinhl	thrd_join	vsscanf
remquof	sinl	thrd_sleep	vswprintf
remquol	snprintf	thrd_yield	vswscanf
rename	sprintf	time	vwprintf
rewind	sqrt	timespec_get	vwscanf
rint	sqrtf	tmpfile	wcrtomb
rintf	sqrtl	tmpnam	wcs[a-z]*
rintl	srand	to[a-z]*	wctob
round	sscanf	trunc	wctomb
roundf	str[a-z]*	truncf	wctrans
roundl	swprintf	truncl	wctype
scalbln	swscanf	tss_create	wmemchr
scalblnf	system	tss_delete	wmemcmp
scalblnl	tan	tss_get	wmemcpy
scalbn	tanf	tss_set	wmemmove
scalbnf	tanh	ungetc	wmemset
scalbnl	tanhf	ungetwc	wprintf
scanf	tanhl	va_copy	wscanf
setbuf	tanl	va_end	

Note: The notation [a–z] indicates any lowercase letter in the portable character set. The notation `'*'` indicates any sequence of zero or more characters that are valid in identifiers with external linkage.

Applications shall not declare or define identifiers with the same name as an identifier reserved in the same context. Since macro names are replaced whenever found, independent of scope and name space, macro names matching any of the reserved identifier names shall not be defined by an application if any associated header is included.

Except that the effect of each inclusion of **<assert.h>** depends on the definition of NDEBUG, headers may be included in any order, and each may be included more than once in a given scope, with no difference in effect from that of being included only once.

If used, the application shall ensure that a header is included outside of any external declaration or definition, and it shall be first included before the first reference to any type or macro it defines, or to any function or object it declares. However, if an identifier is declared or defined in more than one header, the second and subsequent associated headers may be included after the initial reference to the identifier. Prior to the inclusion of a header, or when any macro defined in the header is expanded, the application shall not define any macros with names lexically identical to symbols defined by that header.

2.3 Error Numbers

Most functions can provide an error number. The means by which each function provides its error numbers is specified in its description.

Some functions provide the error number in a variable accessed through the symbol *errno*, defined by including the **<errno.h>** header. The value of *errno* should only be examined when it is indicated to be valid by a function's return value. No function in this volume of POSIX.1-2024 shall set *errno* to zero. For each thread of a process, the value of *errno* shall not be affected by function calls or assignments to *errno* by other threads.

Some functions return an error number directly as the function value. These functions return a value of zero to indicate success.

If more than one error occurs in processing a function call, any one of the possible errors may be returned, as the order of detection is undefined.

Implementations may support additional errors not included in this list, may generate errors included in this list under circumstances other than those described here, or may contain extensions or limitations that prevent some errors from occurring.

The ERRORS section on each reference page specifies which error conditions shall be detected by all implementations (``shall fail'') and which may be optionally detected by an implementation (``may fail''). If no error condition is detected, the action requested shall be successful. If an error condition is detected, the action requested may have been partially performed, unless otherwise stated.

Implementations may generate error numbers listed here under circumstances other than those described, if and only if all those error conditions can always be treated identically to the error conditions as described in this volume of POSIX.1-2024. Implementations shall not generate a different error number from one required by this volume of POSIX.1-2024 for an error condition described in this volume of POSIX.1-2024, but may generate additional errors unless explicitly disallowed for a particular function.

Each implementation shall document, in the conformance document, situations in which each of the optional conditions defined in POSIX.1-2024 is detected. The conformance document may also contain statements that one or more of the optional error conditions are not detected.

Certain threads-related functions are not allowed to return an error code of [EINTR]. Where this applies it is stated in the ERRORS section on the individual function pages.

The following macro names identify the possible error numbers, in the context of the functions specifically defined in this volume of POSIX.1-2024; these general descriptions are more precisely defined in the ERRORS sections of the functions that return them. Only these macro names should be used in programs, since the actual value of the error number is unspecified. All values listed in this section shall be unique, except as noted below. The values for all these macros shall be found in the **<errno.h>** header defined in the Base Definitions volume of POSIX.1-2024. The actual values are unspecified by this volume of POSIX.1-2024.

[E2BIG]

Argument list too long. The sum of the number of bytes used by the new process image's argument list and environment list is greater than the system-imposed limit of {ARG_MAX} bytes.

or:

Lack of space in an output buffer.

or:

17864 Argument is greater than the system-imposed maximum.

17865 [EACCES]
17866 Permission denied. An attempt was made to access a file in a way forbidden by its file
17867 access permissions.

17868 [EADDRINUSE]
17869 Address in use. The specified address is in use.

17870 [EADDRNOTAVAIL]
17871 Address not available. The specified address is not available from the local system.

17872 [EAFNOSUPPORT]
17873 Address family not supported. The implementation does not support the specified address
17874 family, or the specified address is not a valid address for the address family of the specified
17875 socket.

17876 [EAGAIN]
17877 Resource temporarily unavailable. This is a temporary condition and later calls to the same
17878 routine may complete normally.

17879 [EALREADY]
17880 Connection already in progress. A connection request is already in progress for the specified
17881 socket.

17882 [EBADF]
17883 Bad file descriptor. A file descriptor argument is out of range, refers to no open file, or a
17884 read (write) request is made to a file that is only open for writing (reading).

17885 [EBADMSG]
17886 Bad Message. The implementation has detected a corrupted message.

17887 [EBUSY]
17888 Resource busy. An attempt was made to make use of a system resource that is not currently
17889 available, as it is being used by another process in a manner that would have conflicted
17890 with the request being made by this process.

17891 [ECANCELED]
17892 Operation canceled. The associated asynchronous operation was canceled before
17893 completion.

17894 [ECHILD]
17895 No child process. A *wait*(), *waitid*(), or *waitpid*() function was executed by a process that
17896 had no existing or unwaited-for child process.

17897 [ECONNABORTED]
17898 Connection aborted. The connection has been aborted.

17899 [ECONNREFUSED]
17900 Connection refused. An attempt to connect to a socket was refused because there was no
17901 process listening or because the queue of connection requests was full and the underlying
17902 protocol does not support retransmissions.

17903 [ECONNRESET]
17904 Connection reset. The connection was forcibly closed by the peer.

17905 [EDEADLK]
17906 Resource deadlock would occur. An attempt was made to lock a system resource that would
17907 have resulted in a deadlock situation.

17908 [EDESTADDRREQ]
17909 Destination address required. No bind address was established.

17910 [EDOM]
17911 Domain error. An input argument is outside the defined domain of the mathematical
17912 function (defined in the ISO C standard).

17913 [EDQUOT]
17914 Reserved.

17915 [EEXIST]
17916 File exists. An existing file was mentioned in an inappropriate context; for example, as a
17917 new link name in the *link*( ) function.

17918 [EFAULT]
17919 Bad address. The system detected an invalid address in attempting to use an argument of a
17920 call. The reliable detection of this error cannot be guaranteed, and when not detected may
17921 result in the generation of a signal, indicating an address violation, which is sent to the
17922 process.

17923 [EFBIG]
17924 File too large. The size of a file would exceed the implementation's maximum file size, the
17925 file size limit of the process, or the offset maximum established in the corresponding open
17926 file description.

17927 [EHOSTUNREACH]
17928 Host is unreachable. The destination host cannot be reached (probably because the host is
17929 down or a remote router cannot reach it).

17930 [EIDRM]
17931 Identifier removed. Returned during XSI interprocess communication if an identifier has
17932 been removed from the system.

17933 [EILSEQ]
17934 Illegal byte sequence. A wide-character code has been detected that does not correspond to
17935 a valid character, or a byte sequence does not form a valid wide-character code (defined in
17936 the ISO C standard).

17937 [EINPROGRESS]
17938 Operation in progress. This code is used to indicate that an asynchronous operation has not
17939 yet completed.

17940 or:

17941 O_NONBLOCK is set for the socket file descriptor and the connection cannot be
17942 immediately established.

17943 [EINTR]
17944 Interrupted function call. An asynchronous signal was caught by the process during the
17945 execution of an interruptible function. If the signal handler performs a normal return, the
17946 interrupted function call may return this condition (see the Base Definitions volume of
17947 POSIX.1-2024, **<signal.h>**).

17948 [EINVAL]
17949 Invalid argument. Some invalid argument was supplied; for example, specifying an
17950 undefined signal in a *signal*( ) function or a *kill*( ) function.

17951 [EIO]
17952 Input/output error. Some physical input or output error has occurred. This error may be
17953 reported on a subsequent operation on the same file descriptor. Any other error-causing

17954 operation on the same file descriptor may cause the [EIO] error indication to be lost.

17955 [EISCONN]
17956 Socket is connected. The specified socket is already connected.

17957 [EISDIR]
17958 Is a directory. An attempt was made to open a directory with write mode specified.

17959 [ELOOP]
17960 Symbolic link loop. A loop exists in symbolic links encountered during pathname
17961 resolution. This error may also be returned if more than {SYMLOOP_MAX} symbolic links
17962 are encountered during pathname resolution.

17963 [EMFILE]
17964 File descriptor value too large or too many open streams. An attempt was made to open a
17965 XSI file descriptor with a value greater than or equal to {OPEN_MAX}, or greater than or equal
17966 to the soft limit RLIMIT_NOFILE for the process (if smaller than {OPEN_MAX}); or an
17967 attempt was made to open more than the maximum number of streams allowed in the
17968 process.

17969 [EMLINK]
17970 Too many hard links. An attempt was made to have the link count of a single file exceed
17971 {LINK_MAX}.

17972 [EMSGSIZE]
17973 Message too large. A message sent on a transport provider was larger than an internal
17974 message buffer or some other network limit.

17975 or:

17976 Inappropriate message buffer length.

17977 [EMULTIHOP]
17978 Reserved.

17979 [ENAMETOOLONG]
17980 Filename too long. The length of a pathname exceeds {PATH_MAX} and the
17981 implementation considers this to be an error, or a pathname component is longer than
17982 {NAME_MAX}. This error may also occur when pathname substitution, as a result of
17983 encountering a symbolic link during pathname resolution, results in a pathname string the
17984 size of which exceeds {PATH_MAX}.

17985 [ENETDOWN]
17986 Network is down. The local network interface used to reach the destination is down.

17987 [ENETRESET]
17988 The connection was aborted by the network.

17989 [ENETUNREACH]
17990 Network unreachable. No route to the network is present.

17991 [ENFILE]
17992 Too many files open in system. Too many files are currently open in the system. The system
17993 has reached its predefined limit for simultaneously open files and temporarily cannot
17994 accept requests to open another one.

17995 [ENOBUFS]
17996 No buffer space available. Insufficient buffer resources were available in the system to
17997 perform the socket operation.

17998 [ENODEV]
17999 No such device. An attempt was made to apply an inappropriate function to a device; for
18000 example, trying to read a write-only device such as a printer.

18001 [ENOENT]
18002 No such file or directory. A component of a specified pathname does not exist, or the
18003 pathname is an empty string.

18004 [ENOEXEC]
18005 Executable file format error. A request is made to execute a file that, although it has
18006 appropriate privileges, is not in the format required by the implementation for executable
18007 files.

18008 [ENOLCK]
18009 No locks available. A system-imposed limit on the number of simultaneous file and record
18010 locks has been reached and no more are currently available.

18011 [ENOLINK]
18012 Reserved.

18013 [ENOMEM]
18014 Not enough space. The new process image requires more memory than is allowed by the
18015 hardware or system-imposed memory management constraints.

18016 [ENOMSG]
18017 No message of the desired type. The message queue does not contain a message of the
18018 required type during XSI interprocess communication.

18019 [ENOPROTOOPT]
18020 Protocol not available. The protocol option specified to *setsockopt*( ) is not supported by the
18021 implementation.

18022 [ENOSPC]
18023 No space left on a device. During the *write*( ) function on a regular file or when extending a
18024 directory, there is no free space left on the device.

18025 [ENOSYS]
18026 Functionality not supported. An attempt was made to use optional functionality that is not
18027 supported in this implementation.

18028 [ENOTCONN]
18029 Socket not connected. The socket is not connected.

18030 [ENOTDIR]
18031 Not a directory. A component of the specified pathname exists, but it is not a directory,
18032 when a directory was expected; or an attempt was made to create a non-directory file, and
18033 the specified pathname contains at least one non-<slash> character and ends with one or
18034 more trailing <slash> characters.

18035 [ENOTEMPTY]
18036 Directory not empty. A directory other than an empty directory was supplied when an
18037 empty directory was expected.

18038 [ENOTRECOVERABLE]
18039 State not recoverable. The state protected by a robust mutex is not recoverable.

18040 [ENOTSOCK]
18041 Not a socket. The file descriptor does not refer to a socket.

[ENOTSUP]
Not supported. The implementation does not support the requested feature or value.

[ENOTTY]
Inappropriate I/O control operation. A control function has been attempted for a file or special file for which the operation is inappropriate.

[ENXIO]
No such device or address. Input or output on a special file refers to a device that does not exist, or makes a request beyond the capabilities of the device. It may also occur when, for example, a tape drive is not on-line.

[EOPNOTSUPP]
Operation not supported on socket. The type of socket (address family or protocol) does not support the requested operation. A conforming implementation may assign the same values for [EOPNOTSUPP] and [ENOTSUP].

[EOVERFLOW]
Value too large to be stored in data type. An operation was attempted which would generate a value that is outside the range of values that can be represented in the relevant data type or that are allowed for a given data item.

[EOWNERDEAD]
Previous owner died. The owner of a robust mutex terminated while holding the mutex lock.

[EPERM]
Operation not permitted. An attempt was made to perform an operation limited to processes with appropriate privileges or to the owner of a file or other resource.

[EPIPE]
Broken pipe. A write was attempted on a socket, pipe, or FIFO for which there is no process to read the data.

[EPROTO]
Protocol error. Some protocol error occurred. This error is device-specific, but is generally not related to a hardware failure.

[EPROTONOSUPPORT]
Protocol not supported. The protocol is not supported by the address family, or the protocol is not supported by the implementation.

[EPROTOTYPE]
Protocol wrong type for socket. The socket type is not supported by the protocol.

[ERANGE]
Result too large or too small. The result of the function is too large (overflow) or too small (underflow) to be represented in the available space (defined in the ISO C standard).

[EROFS]
Read-only file system. An attempt was made to modify a file or directory on a file system that is read-only.

[ESOCKTNOSUPPORT]
Socket type not supported. The socket type is not supported by the address family, or the socket type is not supported by the implementation.

[ESPIPE]
Invalid seek. An attempt was made to access the file offset associated with a pipe or FIFO.

[ESRCH]
No such process. No process can be found corresponding to that specified by the given process ID.

[ESTALE]
Reserved.

[ETIMEDOUT]
Connection timed out. The connection to a remote machine has timed out. If the connection timed out during execution of the function that reported this error (as opposed to timing out prior to the function being called), it is unspecified whether the function has completed some or all of the documented behavior associated with a successful completion of the function.

or:

Operation timed out. The time limit associated with the operation was exceeded before the operation completed.

[ETXTBSY]
Text file busy. An attempt was made to execute a pure-procedure program that is currently open for writing, or an attempt has been made to open for writing a pure-procedure program that is being executed.

[EWOULDBLOCK]
Operation would block. An operation on a socket marked as non-blocking has encountered a situation such as no data available that otherwise would have caused the function to suspend execution.

A conforming implementation may assign the same values for [EWOULDBLOCK] and [EAGAIN].

[EXDEV]
Improper hard link. Creation of a hard link to a file on another file system was attempted.

2.3.1 Additional Error Numbers

Additional implementation-defined error numbers may be defined in **<errno.h>**.

2.4 Signal Concepts

2.4.1 Signal Generation and Delivery

A signal is said to be ``generated'' for (or sent to) a process or thread when the event that causes the signal first occurs. Examples of such events include detection of hardware faults, timer expiration, signals generated via the **sigevent** structure and terminal activity, as well as invocations of the *kill*() and *sigqueue*() functions. In some circumstances, the same event generates signals for multiple processes.

At the time of generation, a determination shall be made whether the signal has been generated for the process or for a specific thread within the process. Signals which are generated by some action attributable to a particular thread, such as a hardware fault, shall be generated for the thread that caused the signal to be generated. Signals that are generated in association with a

process ID or process group ID or an asynchronous event, such as terminal activity, shall be generated for the process.

Each process has an action to be taken in response to each signal defined by the system (see Section 2.4.3). A signal is said to be ``delivered'' to a process when the appropriate action for the process and signal is taken. A signal is said to be ``accepted'' by a process when the signal is selected and returned by one of the *sigwait*( ) functions.

During the time between the generation of a signal and its delivery or acceptance, the signal is said to be ``pending''. Ordinarily, this interval cannot be detected by an application. However, a signal can be ``blocked'' from delivery to a thread. If the action associated with a blocked signal is anything other than to ignore the signal, and if that signal is generated for the thread, the signal shall remain pending until it is unblocked, it is accepted when it is selected and returned by a call to the *sigwait*( ) function, or the action associated with it is set to ignore the signal. Signals generated for the process shall be delivered to exactly one of those threads within the process which is in a call to a *sigwait*( ) function selecting that signal or has not blocked delivery of the signal. If there are no threads in a call to a *sigwait*( ) function selecting that signal, and if all threads within the process block delivery of the signal, the signal shall remain pending on the process until a thread calls a *sigwait*( ) function selecting that signal, a thread unblocks delivery of the signal, or the action associated with the signal is set to ignore the signal. If the action associated with a blocked signal is to ignore the signal and if that signal is generated for the process, it is unspecified whether the signal is discarded immediately upon generation or remains pending.

Each thread has a ``signal mask'' that defines the set of signals currently blocked from delivery to it. The signal mask for a thread shall be initialized from that of its parent or creating thread, or from the corresponding thread in the parent process if the thread was created as the result of a call to *fork*( ). The *pthread_sigmask*( ), *sigaction*( ), *sigprocmask*( ), and *sigsuspend*( ) functions control the manipulation of the signal mask.

The determination of which action is taken in response to a signal is made at the time the signal is delivered, allowing for any changes since the time of generation. This determination is independent of the means by which the signal was originally generated. If a subsequent occurrence of a pending signal is generated, it is implementation-defined as to whether the signal is delivered or accepted more than once in circumstances other than those in which queuing is required. The order in which multiple, simultaneously pending signals outside the range SIGRTMIN to SIGRTMAX are delivered to or accepted by a process is unspecified.

When any stop signal (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is generated for a process or thread, all pending SIGCONT signals for that process or any of the threads within that process shall be discarded. Conversely, when SIGCONT is generated for a process or thread, all pending stop signals for that process or any of the threads within that process shall be discarded. When SIGCONT is generated for a process that is stopped, the process shall be continued, even if the SIGCONT signal is ignored by the process or is blocked by all threads within the process and there are no threads in a call to a *sigwait*( ) function selecting SIGCONT. If SIGCONT is blocked by all threads within the process, there are no threads in a call to a *sigwait*( ) function selecting SIGCONT, and SIGCONT is not ignored by the process, the SIGCONT signal shall remain pending on the process until it is either unblocked by a thread or a thread calls a *sigwait*( ) function selecting SIGCONT, or a stop signal is generated for the process or any of the threads within the process.

An implementation shall document any condition not specified by this volume of POSIX.1-2024 under which the implementation generates signals.

2.4.2 Realtime Signal Generation and Delivery

This section describes functionality to support realtime signal generation and delivery.

Some signal-generating functions, such as high-resolution timer expiration, asynchronous I/O completion, interprocess message arrival, and the *sigqueue*() function, support the specification of an application-defined value, either explicitly as a parameter to the function or in a **sigevent** structure parameter. The **sigevent** structure is defined in **<signal.h>** and contains at least the following members:

Member Type	Member Name	Description
int	*sigev_notify*	Notification type.
int	*sigev_signo*	Signal number.
union sigval	*sigev_value*	Signal value.
void(*)(union sigval)	*sigev_notify_function*	Notification function.
(pthread_attr_t*)	*sigev_notify_attributes*	Notification attributes.

The *sigev_notify* member specifies the notification mechanism to use when an asynchronous event occurs. This volume of POSIX.1-2024 defines the following values for the *sigev_notify* member:

SIGEV_NONE — No asynchronous notification shall be delivered when the event of interest occurs.

SIGEV_SIGNAL — The signal specified in *sigev_signo* shall be generated for the process when the event of interest occurs. If the SA_SIGINFO flag is set for that signal number, then the signal shall be queued to the process and the value specified in *sigev_value* shall be the *si_value* component of the generated signal. If SA_SIGINFO is not set for that signal number, it is unspecified whether the signal is queued and what value, if any, is sent.

SIGEV_THREAD — A notification function shall be called to perform notification.

An implementation may define additional notification mechanisms.

The *sigev_signo* member specifies the signal to be generated. The *sigev_value* member is the application-defined value to be passed to the signal-catching function at the time of the signal delivery or to be returned at signal acceptance as the *si_value* member of the **siginfo_t** structure.

The **sigval** union is defined in **<signal.h>** and contains at least the following members:

Member Type	Member Name	Description
int	*sival_int*	Integer signal value.
void*	*sival_ptr*	Pointer signal value.

The *sival_int* member shall be used when the application-defined value is of type **int**; the *sival_ptr* member shall be used when the application-defined value is a pointer.

When a signal is generated by the *sigqueue*() function or any signal-generating function that supports the specification of an application-defined value, the signal shall be marked pending and, if the SA_SIGINFO flag is set for that signal, the signal shall be queued to the process along with the application-specified signal value. Multiple occurrences of signals so generated are queued in FIFO order. It is unspecified whether signals so generated are queued when the SA_SIGINFO flag is not set for that signal.

Signals generated by the *kill*() function or other events that cause signals to occur, such as detection of hardware faults, *alarm*() timer expiration, or terminal activity, and for which the implementation does not support queuing, shall have no effect on signals already queued for the

same signal number.

When multiple unblocked signals, all in the range SIGRTMIN to SIGRTMAX, are pending, the behavior shall be as if the implementation delivers the pending unblocked signal with the lowest signal number within that range. No other ordering of signal delivery is specified.

If, when a pending signal is delivered, there are additional signals queued to that signal number, the signal shall remain pending. Otherwise, the pending indication shall be reset.

Multi-threaded programs can use an alternate event notification mechanism. When a notification is processed, and the *sigev_notify* member of the **sigevent** structure has the value SIGEV_THREAD, the function *sigev_notify_function* is called with parameter *sigev_value*.

The function shall be executed in a newly created thread as if it were the *start_routine* for a call to *pthread_create*( ) with the thread attributes specified by *sigev_notify_attributes*. If *sigev_notify_attributes* is NULL, the behavior shall be as if the thread were created with the *detachstate* attribute set to PTHREAD_CREATE_DETACHED. Supplying an attributes structure with a *detachstate* attribute of PTHREAD_CREATE_JOINABLE results in undefined behavior. It is implementation-defined whether the signal mask of this thread has all signals except SIGKILL and SIGSTOP blocked, or is the same as the mask that was in effect for the thread which installed the **sigevent** notification handler at the time of the call that installed the handler.

2.4.3 Signal Actions

There are three types of action that can be associated with a signal: SIG_DFL, SIG_IGN, or a pointer to a function. Initially, all signals shall be set to SIG_DFL or SIG_IGN prior to entry of the *main*( ) routine (see the *exec* functions). The actions prescribed by these values are as follows.

SIG_DFL

Signal-specific default action.

The default actions for the signals defined in this volume of POSIX.1-2024 are specified under **<signal.h>**. The default actions for the realtime signals in the range SIGRTMIN to SIGRTMAX shall be to terminate the process abnormally.

If the default action is to terminate the process abnormally, the process is terminated as if by a call to *_exit*( ), except that the status made available to *wait*( ), *waitid*( ), and *waitpid*( ) indicates abnormal termination by the signal. If the default action is to terminate the process abnormally with additional actions, implementation-defined abnormal termination actions, such as creation of a core image, may also occur.

If the default action is to stop the process, the execution of that process is temporarily suspended. When a process stops, a SIGCHLD signal shall be generated for its parent process, unless the parent process has set the SA_NOCLDSTOP flag. While a process is stopped, any additional signals that are sent to the process shall not be delivered until the process is continued, except SIGKILL which always terminates the receiving process. A process that is a member of an orphaned process group shall not be allowed to stop in response to the SIGTSTP, SIGTTIN, or SIGTTOU signals. In cases where delivery of one of these signals would stop such a process, the signal shall be discarded.

If the default action is to ignore the signal, delivery of the signal shall have no effect on the process.

Setting a signal action to SIG_DFL for a signal that is pending, and whose default action is to ignore the signal (for example, SIGCHLD), shall cause the pending signal to be discarded, whether or not it is blocked. Any queued values pending shall be discarded and the resources

used to queue them shall be released and returned to the system for other use.

The default action for SIGCONT is to resume execution at the point where the process was stopped, after first handling any pending unblocked signals.

XSI When a stopped process is continued, a SIGCHLD signal may be generated for its parent process, unless the parent process has set the SA_NOCLDSTOP flag.

SIG_IGN

Ignore signal.

Delivery of the signal shall have no effect on the process. The behavior of a process is undefined after it ignores a SIGFPE, SIGILL, SIGSEGV, or SIGBUS signal that was not generated by *kill*(), *sigqueue*(), or *raise*().

The system shall not allow the action for the signals SIGKILL or SIGSTOP to be set to SIG_IGN.

Setting a signal action to SIG_IGN for a signal that is pending shall cause the pending signal to be discarded, whether or not it is blocked.

If a process sets the action for the SIGCHLD signal to SIG_IGN, the behavior is unspecified,
XSI except as specified under ``Consequences of Process Termination'' in the description of the *_Exit*() function (see XSH *_Exit*(), on page 568).

Any queued values pending shall be discarded and the resources used to queue them shall be released and made available to queue other signals.

Pointer to a Function

Catch signal.

On delivery of the signal, the receiving process is to execute the signal-catching function at the specified address. After returning from the signal-catching function, the receiving process shall resume execution at the point at which it was interrupted.

If the SA_SIGINFO flag for the signal is cleared, the signal-catching function shall be entered as a C-language function call as follows:

```
void func(int signo);
```

If the SA_SIGINFO flag for the signal is set, the signal-catching function shall be entered as a C-language function call as follows:

```
void func(int signo, siginfo_t *info, void *context);
```

where *func* is the specified signal-catching function, *signo* is the signal number of the signal being delivered, and *info* is a pointer to a **siginfo_t** structure defined in **<signal.h>** containing at least the following members:

Member Type	Member Name	Description
int	*si_signo*	Signal number.
int	*si_code*	Cause of the signal.
pid_t	*si_pid*	Sending process ID.
uid_t	*si_uid*	Real user ID of sending process.
void *	*si_addr*	Address of faulting instruction.
int	*si_status*	Exit value or signal.
union sigval	*si_value*	Signal value.

The *si_signo* member shall contain the signal number. This shall be the same as the *signo* parameter. The *si_code* member shall contain a code identifying the cause of the signal. The

following non-signal-specific values are defined for *si_code*:

SI_USER — The signal was sent by the *kill*( ) function. The implementation may set *si_code* to SI_USER if the signal was sent by the *raise*( ) or *abort*( ) functions or any similar functions provided as implementation extensions.

SI_QUEUE — The signal was sent by the *sigqueue*( ) function.

SI_TIMER — The signal was generated by the expiration of a timer set by *timer_settime*( ).

SI_ASYNCIO — The signal was generated by the completion of an asynchronous I/O request.

MSG SI_MESGQ — The signal was generated by the arrival of a message on an empty message queue.

Signal-specific values for *si_code* are also defined, as described in XBD **<signal.h>**.

If the signal was not generated by one of the functions or events listed above, *si_code* shall be set either to one of the signal-specific values described in XBD **<signal.h>**, or to an implementation-defined value that is not equal to any of the values defined above.

XSI If *si_code* is SI_USER or SI_QUEUE, or any value less than or equal to 0, then the signal was generated by a process and *si_pid* and *si_uid* shall be set to the process ID and the real user ID of the sender, respectively.

In addition, *si_addr*, *si_pid*, *si_status*, and *si_uid* shall be set for certain signal-specific values of *si_code*, as described in XBD **<signal.h>**.

If *si_code* is one of SI_QUEUE, SI_TIMER, SI_ASYNCIO, or SI_MESGQ, then *si_value* shall contain the application-specified signal value. Otherwise, the contents of *si_value* are undefined.

The behavior of a process is undefined after it returns normally from a signal-catching function for a SIGBUS, SIGFPE, SIGILL, or SIGSEGV signal that was not generated by *kill*( ), *sigqueue*( ), or *raise*( ).

The system shall not allow a process to catch the signals SIGKILL and SIGSTOP.

If a process establishes a signal-catching function for the SIGCHLD signal while it has a terminated child process for which it has not waited, it is unspecified whether a SIGCHLD signal is generated to indicate that child process.

If the process is multi-threaded, or if the process is single-threaded and a signal handler is executed other than as the result of:

- The process calling *abort*( ), *raise*( ), *kill*( ), *pthread_kill*( ), or *sigqueue*( ) to generate a signal that is not blocked
- A pending signal being unblocked and being delivered before the call that unblocked it returns

the behavior is undefined if:

- The signal handler refers to any object other than *errno* with static or thread storage duration that is not a lock-free atomic object, and not a non-modifiable object (for example, string literals, objects that were defined with a const-qualified type, and objects in memory that is mapped read-only), other than by assigning a value to an object declared as **volatile sig_atomic_t**, unless the previous modification (if any) to the object happens before the signal handler is called and the return from the signal handler happens before the next modification (if any) to the object.

18344 • The signal handler calls any function or function-like macro defined in this standard other
18345 than one of the functions and macros specified below as being async-signal-safe.

18346 The following table defines a set of functions and function-like macros that shall be async-signal-
18347 safe. Therefore, applications can call them, without restriction, from signal-catching functions.
18348 Note that, although there is no restriction on the calls themselves, for certain functions there are
18349 restrictions on subsequent behavior after the function is called from a signal-catching function
18350 (see *longjmp*( )).

18351 *_Exit*( )	*fstat*( )	*mkdir*( )	*setpgid*( )
18352 *_Fork*( )	*fstatat*( )	*mkdirat*( )	*setregid*( )
18353 *_exit*( )	*fsync*( )	*mkfifo*( )	*setresgid*( )
18354 *abort*( )	*ftruncate*( )	*mkfifoat*( )	*setresuid*( )
18355 *accept*( )	*futimens*( )	*mknod*( )	*setreuid*( )
18356 *accept4*( )	*getegid*( )	*mknodat*( )	*setsid*( )
18357 *access*( )	*geteuid*( )	*ntohl*( )	*setsockopt*( )
18358 *aio_error*( )	*getgid*( )	*ntohs*( )	*setuid*( )
18359 *aio_return*( )	*getgroups*( )	*open*( )	*shutdown*( )
18360 *aio_suspend*( )	*getpeername*( )	*openat*( )	*sig2str*( )
18361 *alarm*( )	*getpgrp*( )	*pause*( )	*sigaction*( )
18362 *be16toh*( )	*getpid*( )	*pipe*( )	*sigaddset*( )
18363 *be32toh*( )	*getppid*( )	*pipe2*( )	*sigdelset*( )
18364 *be64toh*( )	*getresgid*( )	*poll*( )	*sigemptyset*( )
18365 *bind*( )	*getresuid*( )	*posix_close*( )	*sigfillset*( )
18366 *cfgetispeed*( )	*getsockname*( )	*ppoll*( )	*sigismember*( )
18367 *cfgetospeed*( )	*getsockopt*( )	*pread*( )	*siglongjmp*( )
18368 *cfsetispeed*( )	*getuid*( )	*pselect*( )	*signal*( )
18369 *cfsetospeed*( )	*htobe16*( )	*pthread_kill*( )	*sigpending*( )
18370 *chdir*( )	*htobe32*( )	*pthread_self*( )	*sigprocmask*( )
18371 *chmod*( )	*htobe64*( )	*pthread_setcancelstate*( )	*sigqueue*( )
18372 *chown*( )	*htole16*( )	*pthread_sigmask*( )	*sigsuspend*( )
18373 *clock_gettime*( )	*htole32*( )	*pwrite*( )	*sleep*( )
18374 *close*( )	*htole64*( )	*quick_exit*( )	*sockatmark*( )
18375 *connect*( )	*htonl*( )	*raise*( )	*socket*( )
18376 *creat*( )	*htons*( )	*read*( )	*socketpair*( )
18377 *dup*( )	*kill*( )	*readv*( )	*stat*( )
18378 *dup2*( )	*killpg*( )	*readlink*( )	*stpcpy*( )
18379 *dup3*( )	*le16toh*( )	*readlinkat*( )	*stpncpy*( )
18380 *execl*( )	*le32toh*( )	*recv*( )	*strcat*( )
18381 *execle*( )	*le64toh*( )	*recvfrom*( )	*strchr*( )
18382 *execv*( )	*link*( )	*recvmsg*( )	*strcmp*( )
18383 *execve*( )	*linkat*( )	*rename*( )	*strcpy*( )
18384 *faccessat*( )	*listen*( )	*renameat*( )	*strcspn*( )
18385 *fchdir*( )	*longjmp*( )	*rmdir*( )	*strlcat*( )
18386 *fchmod*( )	*lseek*( )	*select*( )	*strlcpy*( )
18387 *fchmodat*( )	*lstat*( )	*sem_post*( )	*strlen*( )
18388 *fchown*( )	*memccpy*( )	*send*( )	*strncat*( )
18389 *fchownat*( )	*memchr*( )	*sendmsg*( )	*strncmp*( )
18390 *fcntl*( )	*memcmp*( )	*sendto*( )	*strncpy*( )
18391 *fdatasync*( )	*memcpy*( )	*setegid*( )	*strnlen*( )
18392 *fexecve*( )	*memmove*( )	*seteuid*( )	*strpbrk*( )
18393 *ffs*( )	*memset*( )	*setgid*( )	*strrchr*( )

strspn( )	*time*( )	*wait*( )	*wcsncpy*( )
strstr( )	*timer_getoverrun*( )	*waitid*( )	*wcsnlen*( )
strtok_r( )	*timer_gettime*( )	*waitpid*( )	*wcspbrk*( )
symlink( )	*timer_settime*( )	*wcpcpy*( )	*wcsrchr*( )
symlinkat( )	*times*( )	*wcpncpy*( )	*wcsspn*( )
tcdrain( )	*umask*( )	*wcscat*( )	*wcsstr*( )
tcflow( )	*uname*( )	*wcschr*( )	*wcstok*( )
tcflush( )	*unlink*( )	*wcscmp*( )	*wmemchr*( )
tcgetattr( )	*unlinkat*( )	*wcscpy*( )	*wmemcmp*( )
tcgetpgrp( )	*utimensat*( )	*wcscspn*( )	*wmemcpy*( )
tcgetwinsize( )	*utimes*( )	*wcslcat*( )	*wmemmove*( )
tcsendbreak( )	*va_arg*( )	*wcslcpy*( )	*wmemset*( )
tcsetattr( )	*va_copy*( )	*wcslen*( )	*write*( )
tcsetpgrp( )	*va_end*( )	*wcsncat*( )	*writev*( )
tcsetwinsize( )	*va_start*( )	*wcsncmp*( )	

In addition, the functions in **<stdatomic.h>** other than *atomic_init*( ) shall be async-signal-safe when the atomic arguments are lock-free, and the *atomic_is_lock_free*( ) function shall be async-signal-safe when called with an atomic argument.

All other functions (including generic functions) and function-like macros may be unsafe with respect to signals. It is implementation-defined which additional interfaces, if any, are also async-signal-safe. In the presence of signals, all functions defined by this volume of POSIX.1-2024 shall behave as defined when called from or interrupted by a signal-catching function, with the exception that when a signal interrupts an unsafe function or function-like macro, or equivalent (such as the processing equivalent to *exit*( ) performed after a return from the initial call to *main*( )), and the signal-catching function calls an unsafe function or function-like macro, the behavior is undefined. Additional exceptions are specified in the descriptions of individual functions such as *longjmp*( ).

Operations which obtain the value of *errno* and operations which assign a value to *errno* shall be async-signal-safe, provided that the signal-catching function saves the value of *errno* upon entry and restores it before it returns.

When a signal is delivered to a thread, if the action of that signal specifies termination, stop, or continue, the entire process shall be terminated, stopped, or continued, respectively.

2.4.4 Signal Effects on Other Functions

Signals affect the behavior of certain functions defined by this volume of POSIX.1-2024 if delivered to a process while it is executing such a function. If the action of the signal is to terminate the process, the process shall be terminated and the function shall not return. If the action of the signal is to stop the process, the process shall stop until continued or terminated. Generation of a SIGCONT signal for the process shall cause the process to be continued, and the original function shall continue at the point the process was stopped. If the action of the signal is to invoke a signal-catching function, the signal-catching function shall be invoked; in this case the original function is said to be ``interrupted'' by the signal. If the signal-catching function executes a **return** statement, the behavior of the interrupted function shall be as described individually for that function, except as noted for unsafe functions. After returning from a signal-catching function, the value of *errno* is unspecified if the signal-catching function or any function it called assigned a value to *errno* and the signal-catching function did not save and restore the original value of *errno*. Signals that are ignored shall not affect the behavior of any function; signals that are blocked shall not affect the behavior of any function until they are unblocked and then delivered, except as specified for the *sigpending*( ) and *sigwait*( ) functions.

2.5 Standard I/O Streams

CX A stream is associated with an external file (which may be a physical device) or memory buffer
CX by ``opening'' a file or buffer. This may involve ``creating'' a new file. Creating an existing file causes its former contents to be discarded if necessary. If a file can support positioning requests (such as a disk file, as opposed to a terminal), then a ``file position indicator'' associated with the stream is positioned at the start (byte number 0) of the file, unless the file is opened with append mode, in which case it is implementation-defined whether the file position indicator is initially positioned at the beginning or end of the file. The file position indicator is maintained by subsequent reads, writes, and positioning requests, to facilitate an orderly progression through the file.

The wide-character input functions shall read characters from the stream and convert them to wide characters as if they were read by successive calls to the *fgetwc*() function. Each conversion shall occur as if by a call to the *mbrtowc*() function, with the conversion state described by the stream's own **mbstate_t** object (see Section 2.5.2, on page 524). The byte input functions shall read characters from the stream as if by successive calls to the *fgetc*() function.

The wide-character output functions shall convert wide characters to characters and write them to the stream as if they were written by successive calls to the *fputwc*() function. Each conversion shall occur as if by a call to the *wcrtomb*() function, with the conversion state described by the stream's own **mbstate_t** object (see Section 2.5.2, on page 524). The byte output functions shall write characters to the stream as if by successive calls to the *fputc*() function.

The *perror*(), *psiginfo*(), and *psignal*() functions shall behave as described above for the byte output functions if the stream is already byte-oriented, and shall behave as described above for the wide-character output functions if the stream is already wide-oriented. If the stream has no orientation, they shall behave as described for the byte output functions except that they shall not change the orientation of the stream.

Functions other than *perror*(), *psiginfo*(), and *psignal*() that write to streams but are neither wide-character output nor byte output functions (*getopt*() and *wordexp*()), shall behave as described above for the byte output functions, except that if the stream has no orientation, it is unspecified whether they set the stream to byte orientation or leave it with no orientation.

When a stream is ``unbuffered'', bytes are intended to appear from the source or at the destination as soon as possible; otherwise, bytes may be accumulated and transmitted as a block. When a stream is ``fully buffered'', bytes are intended to be transmitted as a block when a buffer is filled. When a stream is ``line buffered'', bytes are intended to be transmitted as a block when a <newline> byte is encountered. Furthermore, bytes are intended to be transmitted as a block when a buffer is filled, when input is requested on an unbuffered stream, or when input is requested on a line-buffered stream that requires the transmission of bytes. Support for these characteristics is implementation-defined, and may be affected via *setbuf*() and *setvbuf*().

A file may be disassociated from a controlling stream by ``closing'' the file. Output streams are flushed (any unwritten buffer contents are transmitted) before the stream is disassociated from the file. The value of a pointer to a **FILE** object is unspecified after the associated file is closed (including the standard streams).

A file may be subsequently reopened, by the same or another program execution, and its contents reclaimed or modified (if it can be repositioned at its start). If the *main*() function returns to its original caller, or if the *exit*() function is called, all open files are closed (hence all output streams are flushed) before program termination. Other paths to program termination, such as calling *abort*(), need not close all files properly.

The address of the **FILE** object used to control a stream may be significant; a copy of a **FILE** object need not necessarily serve in place of the original.

At program start-up, three streams shall be predefined and already open: *stdin* (standard input, for conventional input) for reading, *stdout* (standard output, for conventional output) for writing, and *stderr* (standard error, for diagnostic output) for writing. When opened, *stderr* shall not be fully buffered; *stdin* and *stdout* shall be fully buffered if and only if the file descriptor associated with the stream is determined not to be associated with an interactive device.

Each stream shall have an associated lock that is used to prevent data races when multiple threads of execution access a stream, and to restrict the interleaving of stream operations performed by multiple threads. Only one thread can hold this lock at a time. The lock shall be reentrant: a single thread can hold the lock multiple times at a given time. All functions that read, write, position, or query the position of a stream, except those with names ending *_unlocked*, shall lock the stream as if by a call to *flockfile*( ) before accessing it and release the lock as if by a call to *funlockfile*( ) when the access is complete.

If the lock is not immediately available, the function shall wait for it to become available, except in the following circumstances. If the stream is line buffered and is open for writing or for update, and the reason the function is attempting to lock the stream is because it is going to request input on another stream that is unbuffered, or is line buffered and requires the transmission of characters from the host environment (see above), then the function shall attempt to determine whether a deadlock situation exists. If a deadlock situation is found to exist, the function shall fail. If the function is able to establish that a deadlock situation does not exist, it shall wait for the lock to become available. If the function does not establish whether or not a deadlock situation exists, it shall continue as if it had already locked the stream, found its buffer to be empty, and released the lock.

A stream associated with a memory buffer shall have the same operations for text files that a stream associated with an external file would have. In addition, the stream orientation shall be determined in exactly the same fashion.

Input and output operations on a stream associated with a memory buffer by a call to *fmemopen*( ) shall be constrained by the implementation to take place within the bounds of the memory buffer. In the case of a stream opened by *open_memstream*( ) or *open_wmemstream*( ), the memory area shall grow dynamically to accommodate write operations as necessary. For output, if the stream is fully buffered or line buffered, data shall be moved from the stream's internal buffer, or a buffer provided by *setvbuf*( ), to the memory buffer during a flush or close operation. For input, it is unspecified whether a buffer provided by *setvbuf*( ) is used or whether read operations read directly from the memory buffer provided to or allocated by *fmemopen*( ).

When a standard I/O stream has an associated memory buffer (whether allocated internally, supplied to *setvbuf*( ), or supplied to *fmemopen*( )), the behavior is undefined if that buffer overlaps with the destination buffer passed to a call that reads from the stream or with the source buffer passed to a call that writes to the stream.

2.5.1 Interaction of File Descriptors and Standard I/O Streams

This section describes the interaction of file descriptors and standard I/O streams. The functionality described in this section is an extension to the ISO C standard (and the rest of this section is not further CX shaded).

An open file description may be accessed through a file descriptor, which is created using functions such as *open*( ) or *pipe*( ), or through a stream, which is created using functions such as *fopen*( ) or *popen*( ). Either a file descriptor or a stream is called a ``handle'' on the open file description to which it refers; an open file description may have several handles.

Handles can be created or destroyed by explicit user action, without affecting the underlying open file description. Some of the ways to create them include *fcntl*( ), *dup*( ), *fdopen*( ), *fileno*( ),

and *fork*(). They can be destroyed by at least *fclose*(), *close*(), and the *exec* functions.

A file descriptor that is never used in an operation that could affect the file offset (for example, *read*(), *write*(), or *lseek*()) is not considered a handle for this discussion, but could give rise to one (for example, as a consequence of *fdopen*(), *dup*(), or *fork*()). This exception does not include the file descriptor underlying a stream, whether created with *fopen*() or *fdopen*(), so long as it is not used directly by the application to affect the file offset. The *read*() and *write*() functions implicitly affect the file offset; *lseek*() explicitly affects it.

The result of function calls involving any one handle (the ``active handle'') is defined elsewhere in this volume of POSIX.1-2024, but if two or more handles are used, and any one of them is a stream, the application shall ensure that their actions are coordinated as described below. If this is not done, the result is undefined.

A handle which is a stream is considered to be closed when either an *fclose*(), or *freopen*() with non-null filename, is executed on it (for *freopen*() with a null filename, it is implementation-defined whether a new handle is created or the existing one reused), or when the process owning that stream terminates with *exit*(), *abort*(), or due to a signal. Several functions close file descriptors, including *close*(), *dup2*(), *_exit*(), the *exec* functions when FD_CLOEXEC is set on a file descriptor, *fork*() when FD_CLOFORK is set on a file descriptor, and *posix_spawn*() when either FD_CLOEXEC or FD_CLOFORK is set.

For a handle to become the active handle, the application shall ensure that the actions below are performed between the last use of the handle (the current active handle) and the first use of the second handle (the future active handle). The second handle then becomes the active handle. All activity by the application affecting the file offset on the first handle shall be suspended until it again becomes the active file handle. (If a stream function has as an underlying function one that affects the file offset, the stream function shall be considered to affect the file offset.)

The handles need not be in the same process for these rules to apply.

Note that after a *fork*(), two handles exist where one existed before. The application shall ensure that, if both handles can ever be accessed, they are both in a state where the other could become the active handle first. The application shall prepare for a *fork*() exactly as if it were a change of active handle. (If the only action performed by one of the processes is one of the *exec* functions or *_exit*() (not *exit*()), the handle is never accessed in that process.)

For the first handle, the first applicable condition below applies. After the actions required below are taken, if the handle is still open, the application can close it.

- If it is a file descriptor, no action is required.
- If the only further action to be performed on any handle to this open file descriptor is to close it, no action need be taken.
- If it is a stream which is unbuffered, no action need be taken.
- If it is a stream which is line buffered, and the last byte written to the stream was a <newline> (that is, as if a:

  ```
  putc('\n')
  ```

  was the most recent operation on that stream), no action need be taken.
- If it is a stream which is open for writing or appending (but not also open for reading), the application shall either perform an *fflush*(), or the stream shall be closed.

- If the stream is open for reading and it is at the end of the file (*feof*( ) is true), no action need be taken.
- If the stream is open with a mode that allows reading and the underlying open file description refers to a device that is capable of seeking, the application shall either perform an *fflush*( ), or the stream shall be closed.

For the second handle:

- If any previous active handle has been used by a function that explicitly changed the file offset, except as required above for the first handle, the application shall perform an *lseek*( ) or *fseek*( ) (as appropriate to the type of handle) to an appropriate location.

If the active handle ceases to be accessible before the requirements on the first handle, above, have been met, the state of the open file description becomes undefined. This might occur during functions such as a *fork*( ) or *_exit*( ).

The *exec* functions make inaccessible all streams that are open at the time they are called, independent of which streams or file descriptors may be available to the new process image.

When these rules are followed, regardless of the sequence of handles used, no data shall be lost or duplicated when writing, and all data shall be written in order, except as requested by seeks. It is implementation-defined whether, and under what conditions, all input is seen exactly once.

Each function that operates on a stream is said to have zero or more ``underlying functions''. This means that the stream function shares certain traits with the underlying functions, but does not require that there be any relation between the implementations of the stream function and its underlying functions.

2.5.2 Stream Orientation and Encoding Rules

The definition of a stream includes an ``orientation''. After a stream is associated with an external file, but before any operations are performed on it, the stream is without orientation. Once a wide-character input/output function has been applied to a stream without orientation, the stream shall become ``wide-oriented''. Similarly, once a byte input/output function has been applied to a stream without orientation, the stream shall become ``byte-oriented''. Only a call to the *freopen*( ) function or the *fwide*( ) function can otherwise alter the orientation of a stream.

A successful call to *freopen*( ) shall remove any orientation. The three predefined streams *standard input*, *standard output*, and *standard error* shall be unoriented at program start-up.

Byte input/output functions cannot be applied to a wide-oriented stream, and wide-character input/output functions cannot be applied to a byte-oriented stream. The remaining stream operations shall not affect and shall not be affected by a stream's orientation, except for the following additional restriction:

- For wide-oriented streams, after a successful call to a file-positioning function that leaves the file position indicator prior to the end-of-file, a wide-character output function can overwrite a partial character; any file contents beyond the byte(s) written are henceforth undefined.

CX Each wide-oriented stream that was not opened with *open_wmemstream*( ) has an associated **mbstate_t** object that stores the current parse state of the stream. A successful call to *fgetpos*( ) shall store a representation of the value of this **mbstate_t** object as part of the value of the **fpos_t** object. A later successful call to *fsetpos*( ) using the same stored **fpos_t** value shall restore the value of the associated **mbstate_t** object as well as the position within the controlled stream.

Implementations that support multiple encoding rules associate an encoding rule with the

stream. The encoding rule shall be determined by the setting of the *LC_CTYPE* category in the current locale at the time when the stream becomes wide-oriented. As with the stream's orientation, the encoding rule associated with a stream cannot be changed once it has been set, except by a successful call to *freopen*( ) which clears the encoding rule and resets the orientation to unoriented.

Although wide-oriented streams are conceptually sequences of wide characters, the external file CX associated with a wide-oriented stream that was not opened with *open_wmemstream*( ) is a sequence of (possibly multi-byte) characters generalized as follows:

- Multi-byte encodings within files may contain embedded null bytes (unlike multi-byte encodings valid for use internal to the program).
- A file need not begin nor end in the initial shift state.

Moreover, the encodings used for characters may differ among files. Both the nature and choice of such encodings are implementation-defined.

CX On streams that were not opened with *open_wmemstream*( ), the wide-character input functions read characters from the stream and convert them to wide characters as if they were read by successive calls to the *fgetwc*( ) function. Each conversion shall occur as if by a call to the CX *mbrtowc*( ) function, with the conversion state described by the stream's own **mbstate_t** object, except the encoding rule associated with the stream is used instead of the encoding rule implied by the *LC_CTYPE* category of the current locale.

CX On streams that were not opened with *open_wmemstream*( ), the wide-character output functions convert wide characters to (possibly multi-byte) characters and write them to the stream as if they were written by successive calls to the *fputwc*( ) function. Each conversion shall occur as if by a call to the *wcrtomb*( ) function, with the conversion state described by the stream's own CX **mbstate_t** object, except the encoding rule associated with the stream is used instead of the encoding rule implied by the *LC_CTYPE* category of the current locale.

An ``encoding error'' shall occur if the character sequence presented to the underlying *mbrtowc*( ) function does not form a valid (generalized) character, or if the code value passed to the underlying *wcrtomb*( ) function does not correspond to a valid (generalized) character. The wide-character input/output functions and the byte input/output functions store the value of the macro [EILSEQ] in *errno* if and only if an encoding error occurs.

2.6 File Descriptor Allocation

All functions that open one or more file descriptors shall, unless specified otherwise, atomically allocate the lowest numbered available (that is, not already open in the calling process) file descriptor at the time of each allocation. Where a single function allocates two file descriptors (for example, *pipe*( ) or *socketpair*( )), the allocations may be independent and therefore applications should not expect them to have adjacent values or depend on which has the higher value.

2.7 XSI Interprocess Communication

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

The following message passing, semaphore, and shared memory services form an XSI interprocess communication facility. Certain aspects of their operation are common, and are defined as follows.

IPC Functions		
msgctl( )	*semctl*( )	*shmat*( )
msgget( )	*semget*( )	*shmctl*( )
msgrcv( )	*semop*( )	*shmdt*( )
msgsnd( )		*shmget*( )

Another interprocess communication facility is provided by functions in the Realtime Option Group; see Section 2.8 (on page 527).

2.7.1 IPC General Description

Each individual shared memory segment, message queue, and semaphore set shall be identified by a unique positive integer, called, respectively, a shared memory identifier, *shmid*, a semaphore identifier, *semid*, and a message queue identifier, *msqid*. The identifiers shall be returned by calls to *shmget*( ), *semget*( ), and *msgget*( ), respectively.

Associated with each identifier is a data structure which contains data related to the operations which may be or may have been performed; see the Base Definitions volume of POSIX.1-2024, **<sys/shm.h>**, **<sys/sem.h>**, and **<sys/msg.h>** for their descriptions.

Each of the data structures contains both ownership information and an **ipc_perm** structure (see the Base Definitions volume of POSIX.1-2024, **<sys/ipc.h>**) which are used in conjunction to determine whether or not read/write (read/alter for semaphores) permissions should be granted to processes using the IPC facilities. The *mode* member of the **ipc_perm** structure acts as a bit field which determines the permissions.

The values of the bits are given below in octal notation along with the symbolic constants defined in **<sys/stat.h>** that can be used to represent them.

Octal Value	<sys/stat.h> Symbolic Constant	Meaning
0400	S_IRUSR	Read by user.
0200	S_IWUSR	Write (for shared memory & message queues) or alter (for semaphores) by user.
0040	S_IRGRP	Read by group.
0020	S_IWGRP	Write or alter by group.
0004	S_IROTH	Read by others.
0002	S_IWOTH	Write or alter by others.

The name of the **ipc_perm** structure is *shm_perm*, *sem_perm*, or *msg_perm*, depending on which service is being used. In each case, read and write/alter permissions shall be granted to a process if one or more of the following are true (`"xxx"` is replaced by *shm*, *sem*, or *msg*, as appropriate):

- The process has appropriate privileges.
- The effective user ID of the process matches *xxx_perm.cuid* or *xxx_perm.uid* in the data structure associated with the IPC identifier, and the appropriate bit of the *user* field in *xxx_perm.mode* is set.
- The effective user ID of the process does not match *xxx_perm.cuid* or *xxx_perm.uid* but the effective group ID of the process matches *xxx_perm.cgid* or *xxx_perm.gid* in the data structure associated with the IPC identifier, and the appropriate bit of the *group* field in *xxx_perm.mode* is set.
- The effective user ID of the process does not match *xxx_perm.cuid* or *xxx_perm.uid* and the effective group ID of the process does not match *xxx_perm.cgid* or *xxx_perm.gid* in the data structure associated with the IPC identifier, but the appropriate bit of the *other* field in *xxx_perm.mode* is set.

Otherwise, the permission shall be denied.

In addition to the **ipc_perm** structure, each associated data structure includes several **time_t** fields for recording timestamps of particular operations. When an operation is described as setting a timestamp to the current time, that particular timestamp member of the associated data structure shall be set to the largest **time_t** value which is not greater than the current time.

2.8 Realtime

This section defines functions to support the source portability of applications with realtime requirements. The presence of some of these functions is dependent on support for implementation options described in the text.

The specific functional areas included in this section and their scope include the following. Full definitions of these terms can be found in XBD Chapter 3 (on page 31).

- Semaphores
- Process Memory Locking
- Memory Mapped Files and Shared Memory Objects
- Priority Scheduling
- Realtime Signal Extension
- Timers
- Interprocess Communication
- Synchronized Input and Output
- Asynchronous Input and Output

All the realtime functions defined in this volume of POSIX.1-2024 are portable, although some of the numeric parameters used by an implementation may have hardware dependencies.

2.8.1 Realtime Signals

See Section 2.4.2 (on page 515).

2.8.2 Asynchronous I/O

An asynchronous I/O control block structure **aiocb** is used in many asynchronous I/O functions. It is defined in the Base Definitions volume of POSIX.1-2024, **<aio.h>** and has at least the following members:

Member Type	Member Name	Description
int	*aio_fildes*	File descriptor.
off_t	*aio_offset*	File offset.
volatile void*	*aio_buf*	Location of buffer.
size_t	*aio_nbytes*	Length of transfer.
int	*aio_reqprio*	Request priority offset.
struct sigevent	*aio_sigevent*	Signal number and value.
int	*aio_lio_opcode*	Operation to be performed.

The *aio_fildes* element is the file descriptor on which the asynchronous operation is performed.

If O_APPEND is not set for the file descriptor *aio_fildes* and if *aio_fildes* is associated with a device that is capable of seeking, then the requested operation takes place at the absolute position in the file as given by *aio_offset*, as if *lseek*() were called immediately prior to the operation with an *offset* argument equal to *aio_offset* and a *whence* argument equal to SEEK_SET. If O_APPEND is set for the file descriptor, or if *aio_fildes* is associated with a device that is incapable of seeking, write operations append to the file in the same order as the calls were made, with the following exception: under implementation-defined circumstances, such as operation on a multi-processor or when requests of differing priorities are submitted at the same time, the ordering restriction may be relaxed. Since there is no way for a strictly conforming application to determine whether this relaxation applies, all strictly conforming applications which rely on ordering of output shall be written in such a way that they operate correctly if the relaxation applies. After a successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file is unspecified. The *aio_nbytes* and *aio_buf* elements are the same as the *nbyte* and *buf* arguments defined by *read*() and *write*(), respectively.

If _POSIX_PRIORITIZED_IO and _POSIX_PRIORITY_SCHEDULING are defined, then asynchronous I/O is queued in priority order, with the priority of each asynchronous operation based on the current scheduling priority of the calling process. The *aio_reqprio* member can be used to lower (but not raise) the asynchronous I/O operation priority and is within the range zero through {AIO_PRIO_DELTA_MAX}, inclusive. Unless both _POSIX_PRIORITIZED_IO and _POSIX_PRIORITY_SCHEDULING are defined, the order of processing asynchronous I/O requests is unspecified. When both _POSIX_PRIORITIZED_IO and _POSIX_PRIORITY_SCHEDULING are defined, the order of processing of requests submitted by processes whose schedulers are not SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC is unspecified. The priority of an asynchronous request is computed as (process scheduling priority) minus *aio_reqprio*. The priority assigned to each asynchronous I/O request is an indication of the desired order of execution of the request relative to other asynchronous I/O requests for this file. If _POSIX_PRIORITIZED_IO is defined, requests issued with the same priority to a character special file are processed by the underlying device in FIFO order; the order of processing of requests of the same priority issued to files that are not character special files is unspecified. Numerically higher priority values indicate requests of higher priority. The value of *aio_reqprio* has no effect on process scheduling priority. When prioritized asynchronous I/O requests to the same file are blocked waiting for a resource required for that I/O operation,

the higher-priority I/O requests shall be granted the resource before lower-priority I/O requests are granted the resource. The relative priority of asynchronous I/O and synchronous I/O is implementation-defined. If _POSIX_PRIORITIZED_IO is defined, the implementation shall define for which files I/O prioritization is supported.

The *aio_sigevent* determines how the calling process shall be notified upon I/O completion, as specified in Section 2.4.1 (on page 513). If *aio_sigevent.sigev_notify* is SIGEV_NONE, then no signal shall be posted upon I/O completion, but the error status for the operation and the return status for the operation shall be set appropriately.

The *aio_lio_opcode* field is used only by the *lio_listio*( ) call. The *lio_listio*( ) call allows multiple asynchronous I/O operations to be submitted at a single time. The function takes as an argument an array of pointers to **aiocb** structures. Each **aiocb** structure indicates the operation to be performed (read or write) via the *aio_lio_opcode* field.

The address of the **aiocb** structure is used as a handle for retrieving the error status and return status of the asynchronous operation while it is in progress.

The **aiocb** structure and the data buffers associated with the asynchronous I/O operation are being used by the system for asynchronous I/O while, and only while, the error status of the asynchronous operation is equal to [EINPROGRESS]. Applications shall not modify the **aiocb** structure while the structure is being used by the system for asynchronous I/O.

The return status of the asynchronous operation is the number of bytes transferred by the I/O operation. If the error status is set to indicate an error completion, then the return status is set to the return value that the corresponding *read*( ), *write*( ), or *fsync*( ) call would have returned. When the error status is not equal to [EINPROGRESS], the return status shall reflect the return status of the corresponding synchronous operation.

2.8.3 Memory Management

2.8.3.1 *Memory Locking*

MLR Range memory locking operations are defined in terms of pages. Implementations may restrict the size and alignment of range lockings to be on page-size boundaries. The page size, in bytes, is the value of the configurable system variable {PAGESIZE}. If an implementation has no restrictions on size or alignment, it may specify a 1-byte page size.

ML|MLR Memory locking guarantees the residence of portions of the address space. It is implementation-defined whether locking memory guarantees fixed translation between virtual addresses (as seen by the process) and physical addresses. Per-process memory locks are not inherited across a *fork*( ), and all memory locks owned by a process are unlocked upon *exec* or process termination. Unmapping of an address range removes any memory locks established on that address range by this process.

2.8.3.2 *Memory Mapped Files*

Range memory mapping operations are defined in terms of pages. Implementations may restrict the size and alignment of range mappings to be on page-size boundaries. The page size, in bytes, is the value of the configurable system variable {PAGESIZE}. If an implementation has no restrictions on size or alignment, it may specify a 1-byte page size.

Memory mapped files provide a mechanism that allows a process to access files by directly

incorporating file data into its address space. Once a file is mapped into a process address space, the data can be manipulated as memory. If more than one process maps a file, its contents are shared among them. If the mappings allow shared write access, then data written into the memory object through the address space of one process appears in the address spaces of all processes that similarly map the same portion of the memory object.

SHM Shared memory objects are named regions of storage that may be independent of the file system and can be mapped into the address space of one or more processes to allow them to share the associated memory.

SHM An *unlink*( ) of a file or *shm_unlink*( ) of a shared memory object, while causing the removal of the name, does not unmap any mappings established for the object. Once the name has been removed, the contents of the memory object are preserved as long as it is referenced. The memory object remains referenced as long as a process has the memory object open or has some area of the memory object mapped.

2.8.3.3 *Memory Protection*

When an object is mapped, various application accesses to the mapped region may result in signals. In this context, SIGBUS is used to indicate an error using the mapped object, and SIGSEGV is used to indicate a protection violation or misuse of an address:

- A mapping may be restricted to disallow some types of access.
- Write attempts to memory that was mapped without write access, or any access to memory mapped PROT_NONE, shall result in a SIGSEGV signal.
- References to unmapped addresses shall result in a SIGSEGV signal.
- Reference to whole pages within the mapping, but beyond the current length of the object, shall result in a SIGBUS signal.
- The size of the object is unaffected by access beyond the end of the object (even if a SIGBUS is not generated).

2.8.3.4 *Typed Memory Objects*

TYM The functionality described in this section shall be provided on implementations that support the Typed Memory Objects option (and the rest of this section is not further shaded for this option).

Implementations may support the Typed Memory Objects option independently of support for memory mapped files or shared memory objects. Typed memory objects are implementation-configurable named storage pools accessible from one or more processors in a system, each via one or more ports, such as backplane buses, LANs, I/O channels, and so on. Each valid combination of a storage pool and a port is identified through a name that is defined at system configuration time, in an implementation-defined manner; the name may be independent of the file system. Using this name, a typed memory object can be opened and mapped into process address space. For a given storage pool and port, it is necessary to support both dynamic allocation from the pool as well as mapping at an application-supplied offset within the pool; when dynamic allocation has been performed, subsequent deallocation shall be supported. Lastly, accessing typed memory objects from different ports requires a method for obtaining the offset and length of contiguous storage of a region of typed memory (dynamically allocated or not); this allows typed memory to be shared among processes and/or processors while being accessed from the desired port.

2.8.4 Process Scheduling

PS The functionality described in this section shall be provided on implementations that support the Process Scheduling option (and the rest of this section is not further shaded for this option).

Scheduling Policies

The scheduling semantics described in this volume of POSIX.1-2024 are defined in terms of a conceptual model that contains a set of thread lists. No implementation structures are necessarily implied by the use of this conceptual model. It is assumed that no time elapses during operations described using this model, and therefore no simultaneous operations are possible. This model discusses only processor scheduling for runnable threads, but it should be noted that greatly enhanced predictability of realtime applications results if the sequencing of other resources takes processor scheduling policy into account.

There is, conceptually, one thread list for each priority. A runnable thread shall be on the thread list for that thread's priority. Multiple scheduling policies shall be provided. Each non-empty thread list is ordered, contains a head as one end of its order, and a tail as the other. The purpose of a scheduling policy is to define the allowable operations on this set of lists (for example, moving threads between and within lists).

The POSIX model treats a ``process'' as an aggregation of system resources, including one or more threads that may be scheduled by the operating system on the processor(s) it controls. Although a process has its own set of scheduling attributes, these have an indirect effect (if any) on the scheduling behavior of individual threads as described below.

Each thread shall be controlled by an associated scheduling policy and priority. These parameters may be specified by explicit application execution of the *pthread_setschedparam*( ) function. Additionally, the scheduling parameters of a thread (but not its scheduling policy) may be changed by application execution of the *pthread_setschedprio*( ) function.

Each process shall be controlled by an associated scheduling policy and priority. These parameters may be specified by explicit application execution of the *sched_setscheduler*( ) or *sched_setparam*( ) functions.

The effect of the process scheduling attributes on individual threads in the process is dependent on the scheduling contention scope of the threads (see Section 2.9.4, on page 540):

- For threads with system scheduling contention scope, the process scheduling attributes shall have no effect on the scheduling attributes or behavior either of the thread or an underlying kernel scheduling entity dedicated to that thread.
- For threads with process scheduling contention scope, the process scheduling attributes shall have no effect on the scheduling attributes of the thread. However, any underlying kernel scheduling entity used by these threads shall at all times behave as specified by the scheduling attributes of the containing process, and this behavior may affect the scheduling behavior of the process contention scope threads. For example, a process contention scope thread with scheduling policy SCHED_FIFO and the system maximum priority *H* (the value returned by *sched_get_priority_max*(SCHED_FIFO)) in a process with scheduling policy SCHED_RR and system minimum priority *L* (the value returned by *sched_get_priority_min*(SCHED_RR)) shall be subject to timeslicing and to preemption by any thread with an effective priority higher than *L*.

Associated with each policy is a priority range. Each policy definition shall specify the minimum priority range for that policy. The priority ranges for each policy may but need not overlap the priority ranges of other policies.

A conforming implementation shall select the thread that is defined as being at the head of the

highest priority non-empty thread list to become a running thread, regardless of its associated policy. This thread is then removed from its thread list.

Four scheduling policies are specifically required. Other implementation-defined scheduling policies may be defined. The following symbols are defined in the Base Definitions volume of POSIX.1-2024, **<sched.h>**:

SCHED_FIFO First in, first out (FIFO) scheduling policy.

SCHED_RR Round robin scheduling policy.

SS SCHED_SPORADIC Sporadic server scheduling policy.

SCHED_OTHER Another scheduling policy.

The values of these symbols shall be distinct.

SCHED_FIFO

Conforming implementations shall include a scheduling policy called the FIFO scheduling policy.

Threads scheduled under this policy are chosen from a thread list that is ordered by the time its threads have been on the list without being executed; generally, the head of the list is the thread that has been on the list the longest time, and the tail is the thread that has been on the list the shortest time.

Under the SCHED_FIFO policy, the modification of the definitional thread lists is as follows:

1. When a running thread becomes a preempted thread, it becomes the head of the thread list for its priority.
2. When a blocked thread becomes a runnable thread, it becomes the tail of the thread list for its priority.
3. When a running thread calls the *sched_setscheduler*( ) function, the process specified in the function call is modified to the specified policy and the priority specified by the *param* argument.
4. When a running thread calls the *sched_setparam*( ) function, the priority of the process specified in the function call is modified to the priority specified by the *param* argument.
5. When a running thread calls the *pthread_setschedparam*( ) function, the thread specified in the function call is modified to the specified policy and the priority specified by the *param* argument.
6. When a running thread calls the *pthread_setschedprio*( ) function, the thread specified in the function call is modified to the priority specified by the *prio* argument.
7. If a thread whose policy or priority has been modified other than by *pthread_setschedprio*( ) is a running thread or is runnable, it then becomes the tail of the thread list for its new priority.
8. If a thread whose priority has been modified by *pthread_setschedprio*( ) is a running thread or is runnable, the effect on its position in the thread list depends on the direction of the modification, as follows:
    a. If the priority is raised, the thread becomes the tail of the thread list.
    b. If the priority is unchanged, the thread does not change position in the thread list.

18953 c. If the priority is lowered, the thread becomes the head of the thread list.

18954 9. When a running thread issues the *sched_yield*() or *thrd_yield*() function, the thread
18955 becomes the tail of the thread list for its priority.

18956 10. At no other time is the position of a thread with this scheduling policy within the thread
18957 lists affected.

18958 While a thread is executing at a temporarily elevated priority as a consequence of owning a
18959 mutex initialized with the PTHREAD_PRIO_INHERIT or PTHREAD_PRIO_PROTECT protocol
18960 (see *pthread_mutexattr_getprotocol*()), the effects of the above requirements on thread priority
18961 shall apply only to the thread's normal priority, not to its elevated priority, and those of the
18962 above requirements that describe the thread being placed on any thread list as a result of a
18963 priority change shall not apply. Likewise, when such a thread reverts to its normal priority as a
18964 consequence of unlocking such a mutex, those of the above requirements that describe the
18965 thread being placed on any thread list as a result of a priority change shall not apply.

18966 For this policy, valid priorities shall be within the range returned by the *sched_get_priority_max*()
18967 and *sched_get_priority_min*() functions when SCHED_FIFO is provided as the parameter.
18968 Conforming implementations shall provide a priority range of at least 32 priorities for this
18969 policy.

18970 **SCHED_RR**

18971 Conforming implementations shall include a scheduling policy called the ``round robin''
18972 scheduling policy. This policy shall be identical to the SCHED_FIFO policy with the additional
18973 condition that when the implementation detects that a running thread has been executing as a
18974 running thread for a time period of the length returned by the *sched_rr_get_interval*() function or
18975 longer, the thread shall become the tail of its thread list and the head of that thread list shall be
18976 removed and made a running thread.

18977 The effect of this policy is to ensure that if there are multiple SCHED_RR threads at the same
18978 priority, one of them does not monopolize the processor. An application should not rely only on
18979 the use of SCHED_RR to ensure application progress among multiple threads if the application
18980 includes threads using the SCHED_FIFO policy at the same or higher priority levels or
18981 SCHED_RR threads at a higher priority level.

18982 A thread under this policy that is preempted and subsequently resumes execution as a running
18983 thread completes the unexpired portion of its round robin interval time period.

18984 For this policy, valid priorities shall be within the range returned by the *sched_get_priority_max*()
18985 and *sched_get_priority_min*() functions when SCHED_RR is provided as the parameter.
18986 Conforming implementations shall provide a priority range of at least 32 priorities for this
18987 policy.

18988 **SCHED_SPORADIC**

18989 SS | TSP The functionality described in this section shall be provided on implementations that support
18990 the Process Sporadic Server or Thread Sporadic Server options (and the rest of this section is not
18991 further shaded for these options).

18992 If _POSIX_SPORADIC_SERVER or _POSIX_THREAD_SPORADIC_SERVER is defined, the
18993 implementation shall include a scheduling policy identified by the value SCHED_SPORADIC.

18994 The sporadic server policy is based primarily on two time parameters: the replenishment period
18995 and the available execution capacity. The replenishment period is given by the
18996 *sched_ss_repl_period* member of the **sched_param** structure. The available execution capacity is
18997 initialized to the value given by the *sched_ss_init_budget* member of the same parameter. The

sporadic server policy is identical to the SCHED_FIFO policy with some additional conditions that cause the thread's assigned priority to be switched between the values specified by the *sched_priority* and *sched_ss_low_priority* members of the **sched_param** structure.

The priority assigned to a thread using the sporadic server scheduling policy is determined in the following manner: if the available execution capacity is greater than zero and the number of pending replenishment operations is strictly less than *sched_ss_max_repl*, the thread is assigned the priority specified by *sched_priority*; otherwise, the assigned priority shall be *sched_ss_low_priority*. If the value of *sched_priority* is less than or equal to the value of *sched_ss_low_priority*, the results are undefined. When active, the thread shall belong to the thread list corresponding to its assigned priority level, according to the mentioned priority assignment. The modification of the available execution capacity and, consequently of the assigned priority, is done as follows:

1. When the thread at the head of the *sched_priority* list becomes a running thread, its execution time shall be limited to at most its available execution capacity, plus the resolution of the execution time clock used for this scheduling policy. This resolution shall be implementation-defined.

2. Each time the thread is inserted at the tail of the list associated with *sched_priority*—because as a blocked thread it became runnable with priority *sched_priority* or because a replenishment operation was performed—the time at which this operation is done is posted as the *activation_time*.

3. When the running thread with assigned priority equal to *sched_priority* becomes a preempted thread, it becomes the head of the thread list for its priority, and the execution time consumed is subtracted from the available execution capacity. If the available execution capacity would become negative by this operation, it shall be set to zero.

4. When the running thread with assigned priority equal to *sched_priority* becomes a blocked thread, the execution time consumed is subtracted from the available execution capacity, and a replenishment operation is scheduled, as described in 6 and 7. If the available execution capacity would become negative by this operation, it shall be set to zero.

5. When the running thread with assigned priority equal to *sched_priority* reaches the limit imposed on its execution time, it becomes the tail of the thread list for *sched_ss_low_priority*, the execution time consumed is subtracted from the available execution capacity (which becomes zero), and a replenishment operation is scheduled, as described in 6 and 7.

6. Each time a replenishment operation is scheduled, the amount of execution capacity to be replenished, *replenish_amount*, is set equal to the execution time consumed by the thread since the *activation_time*. The replenishment is scheduled to occur at *activation_time* plus *sched_ss_repl_period*. If the scheduled time obtained is before the current time, the replenishment operation is carried out immediately. Several replenishment operations can be pending at the same time, each of which shall be serviced at its respective scheduled time. With the above rules, the number of replenishment operations simultaneously pending for a given thread that is scheduled under the sporadic server policy shall not be greater than *sched_ss_max_repl*.

7. A replenishment operation consists of adding the corresponding *replenish_amount* to the available execution capacity at the scheduled time. If, as a consequence of this operation, the execution capacity would become larger than *sched_ss_initial_budget*, it shall be rounded down to a value equal to *sched_ss_initial_budget*. Additionally, if the thread was runnable or running, and had assigned priority equal to *sched_ss_low_priority*, then it becomes the tail of the thread list for *sched_priority*.

Execution time is defined in XBD Section 3.90 (on page 44).

For this policy, changing the value of a CPU-time clock via *clock_settime*() shall have no effect on its behavior.

For this policy, valid priorities shall be within the range returned by the *sched_get_priority_min*() and *sched_get_priority_max*() functions when SCHED_SPORADIC is provided as the parameter. Conforming implementations shall provide a priority range of at least 32 distinct priorities for this policy.

If the scheduling policy of the target process is either SCHED_FIFO or SCHED_RR, the *sched_ss_low_priority*, *sched_ss_repl_period*, and *sched_ss_init* budget members of the *param* argument shall have no effect on the scheduling behavior. If the scheduling policy of this process is not SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC, the effects of these members are implementation-defined; this case includes the SCHED_OTHER policy.

SCHED_OTHER

Conforming implementations shall include one scheduling policy identified as SCHED_OTHER (which may execute identically with either the FIFO or round robin scheduling policy). The effect of scheduling threads with the SCHED_OTHER policy in a system in which other threads SS are executing under SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC is implementation-defined.

This policy is defined to allow strictly conforming applications to be able to indicate in a portable manner that they no longer need a realtime scheduling policy.

For threads executing under this policy, the implementation shall use only priorities within the range returned by the *sched_get_priority_max*() and *sched_get_priority_min*() functions when SCHED_OTHER is provided as the parameter.

2.8.5 Clocks and Timers

The **<time.h>** header defines the types and manifest constants used by the timing facility.

Time Value Specification Structures

Many of the timing facility functions accept or return time value specifications. A time value structure **timespec** specifies a single time value and includes at least the following members:

Member Type	Member Name	Description
time_t	*tv_sec*	Seconds.
long	*tv_nsec*	Nanoseconds.

The *tv_nsec* member is only valid if greater than or equal to zero, and less than the number of nanoseconds in a second (1 000 million). The time interval described by this structure is (*tv_sec* * 10^9 + *tv_nsec*) nanoseconds.

A time value structure **itimerspec** specifies an initial timer value and a repetition interval for use by the per-process timer functions. This structure includes at least the following members:

Member Type	Member Name	Description
struct timespec	*it_interval*	Timer period.
struct timespec	*it_value*	Timer expiration.

If the value described by *it_value* is non-zero, it indicates the time to or time of the next timer expiration (for relative and absolute timer values, respectively). If the value described by *it_value*

is zero, the timer shall be disarmed.

If the value described by *it_interval* is non-zero, it specifies an interval which shall be used in reloading the timer when it expires; that is, a periodic timer is specified. If the value described by *it_interval* is zero, the timer is disarmed after its next expiration; that is, a one-shot timer is specified.

Timer Event Notification Control Block

Per-process timers may be created that notify the process of timer expirations by queuing a realtime extended signal. The **sigevent** structure, defined in the Base Definitions volume of POSIX.1-2024, **<signal.h>**, is used in creating such a timer. The **sigevent** structure contains the signal number and an application-specific data value which shall be used when notifying the calling process of timer expiration events.

Manifest Constants

The following constants are defined in the Base Definitions volume of POSIX.1-2024, **<time.h>**:

CLOCK_REALTIME	The identifier for the system-wide realtime clock.
TIMER_ABSTIME	Flag indicating time is absolute with respect to the clock associated with a timer.
CLOCK_MONOTONIC	The identifier for the system-wide monotonic clock, which is defined as a clock whose value cannot be set via *clock_settime*( ) and which cannot have backward clock jumps. The maximum possible clock jump is implementation-defined.

The maximum allowable resolution for CLOCK_REALTIME and CLOCK_MONOTONIC clocks and all time services based on these clocks is represented by {_POSIX_CLOCKRES_MIN} and shall be defined as 20 ms (1/50 of a second). Implementations may support smaller values of resolution for these clocks to provide finer granularity time bases. The actual resolution supported by an implementation for a specific clock is obtained using the *clock_getres*( ) function. If the actual resolution supported for a time service based on one of these clocks differs from the resolution supported for that clock, the implementation shall document this difference.

The minimum allowable maximum value for CLOCK_REALTIME and CLOCK_MONOTONIC clocks and all absolute time services based on them is the same as that defined by the ISO C standard for the **time_t** type. If the maximum value supported by a time service based on one of these clocks differs from the maximum value supported by that clock, the implementation shall document this difference.

Execution Time Monitoring

CPT If _POSIX_CPUTIME is defined, process CPU-time clocks shall be supported in addition to the clocks described in Manifest Constants.

TCT If _POSIX_THREAD_CPUTIME is defined, thread CPU-time clocks shall be supported.

CPT|TCT CPU-time clocks measure execution or CPU time, which is defined in XBD Section 3.90 (on page 44). The mechanism used to measure execution time is described in XBD Section 4.14 (on page 99).

CPT If _POSIX_CPUTIME is defined, the following constant of the type **clockid_t** is defined in **<time.h>**:

CLOCK_PROCESS_CPUTIME_ID
When this value of the type **clockid_t** is used in a *clock*() or *timer**() function call, it is interpreted as the identifier of the CPU-time clock associated with the process making the function call.

TCT If _POSIX_THREAD_CPUTIME is defined, the following constant of the type **clockid_t** is defined in **<time.h>**:

CLOCK_THREAD_CPUTIME_ID
When this value of the type **clockid_t** is used in a *clock*() or *timer**() function call, it is interpreted as the identifier of the CPU-time clock associated with the thread making the function call.

2.9 Threads

This section defines functionality to support multiple flows of control, called ``threads'', within a process. For the definition of threads, see XBD Section 3.388 (on page 88).

The specific functional areas covered by threads and their scope include:

- Thread management: the creation, control, and termination of multiple flows of control in the same process under the assumption of a common shared address space
- Synchronization primitives optimized for tightly coupled operation of multiple control flows in a common, shared address space

2.9.1 Thread-Safety

All functions defined by this volume of POSIX.1-2024 shall be thread-safe, except that the following functions[8] need not be thread-safe.

8. The functions in the table are not shaded to denote applicable options. Individual reference pages should be consulted.

asctime( )	*endutxent*( )	*getservent*( )	*putenv*( )
atomic_init( )	*getdate*( )	*getutxent*( )	*pututxline*( )
catgets( )	*getgrent*( )	*getutxid*( )	*rand*( )
crypt( )	*getgrgid*( )	*getutxline*( )	*setenv*( )
ctime( )	*getgrnam*( )	*gmtime*( )	*setgrent*( )
dbm_clearerr( )	*gethostent*( )	*hcreate*( )	*setkey*( )
dbm_close( )	*getlogin*( )	*hdestroy*( )	*setlocale*( )
dbm_delete( )	*getnetbyaddr*( )	*hsearch*( )	*setpwent*( )
dbm_error( )	*getnetbyname*( )	*inet_ntoa*( )	*setutxent*( )
dbm_fetch( )	*getnetent*( )	*l64a*( )	*srand*( )
dbm_firstkey( )	*getopt*( )	*localeconv*( )	*strerror*( )
dbm_nextkey( )	*getprotobyname*( )	*localtime*( )	*strsignal*( )
dbm_open( )	*getprotobynumber*( )	*lrand48*( )	*strtok*( )
dbm_store( )	*getprotoent*( )	*mblen*( )	*ttyname*( )
dlerror( )	*getpwent*( )	*mbtowc*( )	*unsetenv*( )
drand48( )	*getpwnam*( )	*mrand48*( )	*wctomb*( )
encrypt( )	*getpwuid*( )	*nftw*( )	
endgrent( )	*getservbyname*( )	*nl_langinfo*( )	
endpwent( )	*getservbyport*( )	*ptsname*( )	

The *ctermid*( ) and *tmpnam*( ) functions need not be thread-safe if passed a null pointer argument. The *c16rtomb*( ), *c32rtomb*( ), *mbrlen*( ), *mbrtoc16*( ), *mbrtoc32*( ), *mbrtowc*( ), *mbsnrtowcs*( ), *mbsrtowcs*( ), *wcrtomb*( ), *wcsnrtombs*( ), and *wcsrtombs*( ) functions need not be thread-safe if passed a null *ps* argument. The *lgamma*( ), *lgammaf*( ), and *lgammal*( ) functions shall be thread-safe XSI except that they need not avoid data races when storing a value in the *signgam* variable. The *getc_unlocked*( ), *getchar_unlocked*( ), *putc_unlocked*( ), and *putchar_unlocked*( ) functions need not be thread-safe unless the invoking thread owns the (**FILE** *) object accessed by the call, as is the case after a successful call to the *flockfile*( ) or *ftrylockfile*( ) functions. The *readdir*( ) function need not be thread-safe if concurrent calls are made for the same directory stream.

Some functions that are not required to be thread-safe are nevertheless required to avoid data races with either all or some other functions, as specified on their individual reference pages.

Implementations shall provide internal synchronization as necessary in order to satisfy thread-safety requirements.

Since multi-threaded applications are not allowed to use the *environ* variable to access or modify any environment variable while any other thread is concurrently modifying any environment variable, the *getenv*( ) and *secure_getenv*( ) functions and any function dependent on any environment variable are not thread-safe if another thread modifies the environment; see XSH *exec* (on page 866).

2.9.2 Thread IDs

Although implementations may have thread IDs that are unique in a system, applications should only assume that thread IDs are usable and unique within a single process. The effect of calling any of the functions defined in this volume of POSIX.1-2024 and passing as an argument the thread ID of a thread from another process is unspecified. The lifetime of a thread ID ends after the later of thread termination (see Section 3.392, on page 89) and the point when the thread is no longer joinable (see Section 3.183, on page 58). A conforming implementation is free to reuse a thread ID after its lifetime has ended. If an application attempts to use a thread ID whose lifetime has ended, the behavior is undefined.

If a thread is detached, its thread ID is invalid for use as an argument in a call to *pthread_detach*( ), *pthread_join*( ), *thrd_detach*( ), or *thrd_join*( ).

2.9.3 Thread Mutexes

A thread that has blocked shall not prevent any unblocked thread that is eligible to use the same processing resources from eventually making forward progress in its execution. Eligibility for processing resources is determined by the scheduling policy.

A thread shall become the owner of a mutex, *m*, of type **pthread_mutex_t** when one of the following occurs:

- It calls *pthread_mutex_clocklock*( ), *pthread_mutex_lock*( ), *pthread_mutex_timedlock*( ), or *pthread_mutex_trylock*( ) with *m* as the *mutex* argument and the call returns zero or [EOWNERDEAD].
- It calls *pthread_mutex_setprioceiling*( ) with *m* as the *mutex* argument and the call returns [EOWNERDEAD].
- It calls *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), or *pthread_cond_wait*( ) with *m* as the *mutex* argument and the call returns zero or certain error numbers (see *pthread_cond_clockwait*( )).

The thread shall remain the owner of *m* until one of the following occurs:

- It executes *pthread_mutex_unlock*( ) with *m* as the *mutex* argument
- It blocks in a call to *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), or *pthread_cond_wait*( ) with *m* as the *mutex* argument.

A thread shall become the owner of a mutex, *m*, of type **mtx_t** when one of the following occurs:

- It calls *mtx_lock*( ) with *m* as the *mtx* argument and the call returns `thrd_success`.
- It calls *mtx_trylock*( ) with *m* as the *mtx* argument and the call returns `thrd_success`.
- It calls *mtx_timedlock*( ) with *m* as the *mtx* argument and the call returns `thrd_success`.
- It calls *cnd_wait*( ) with *m* as the *mtx* argument and the call returns `thrd_success`.
- It calls *cnd_timedwait*( ) with *m* as the *mtx* argument and the call returns `thrd_success` or `thrd_timedout`.

The thread shall remain the owner of *m* until one of the following occurs:

- It executes *mtx_unlock*( ) with *m* as the *mtx* argument.
- It blocks in a call to *cnd_wait*( ) with *m* as the *mtx* argument.
- It blocks in a call to *cnd_timedwait*( ) with *m* as the *mtx* argument.

The implementation shall behave as if at all times there is at most one owner of any mutex.

A thread that becomes the owner of a mutex is said to have ``acquired'' the mutex and the mutex is said to have become ``locked''; when a thread gives up ownership of a mutex it is said to have ``released'' the mutex and the mutex is said to have become ``unlocked''.

A problem can occur if a process terminates while one of its threads holds a mutex lock. Depending on the mutex type, it might be possible for another thread to unlock the mutex and recover the state of the mutex. However, it is difficult to perform this recovery reliably.

Robust mutexes provide a means to enable the implementation to notify other threads in the event of a process terminating while one of its threads holds a lock on a mutex of type **pthread_mutex_t**. The next thread that acquires the mutex is notified about the termination by the return value [EOWNERDEAD] from the locking function. The notified thread can then attempt to recover the state protected by the mutex, and if successful mark the state protected by the mutex as consistent by a call to *pthread_mutex_consistent*( ). If the notified thread is unable to

recover the state, it can declare the state as not recoverable by a call to *pthread_mutex_unlock*( ) without a prior call to *pthread_mutex_consistent*( ).

Whether or not the state protected by a mutex can be recovered is dependent solely on the application using robust mutexes. The robust mutex support provided in the implementation provides notification only that a mutex owner has terminated while holding a lock, or that the state of the mutex is not recoverable.

2.9.4 Thread Scheduling

TPS The functionality described in this section shall be provided on implementations that support the Thread Execution Scheduling option (and the rest of this section is not further shaded for this option).

Thread Scheduling Attributes

In support of the scheduling function, threads have attributes which are accessed through the **pthread_attr_t** thread creation attributes object.

The *contentionscope* attribute defines the scheduling contention scope of the thread to be either PTHREAD_SCOPE_PROCESS or PTHREAD_SCOPE_SYSTEM.

The *inheritsched* attribute specifies whether a newly created thread is to inherit the scheduling attributes of the creating thread or to have its scheduling values set according to the other scheduling attributes in the **pthread_attr_t** object.

The *schedpolicy* attribute defines the scheduling policy for the thread. The *schedparam* attribute defines the scheduling parameters for the thread. The interaction of threads having different policies within a process is described as part of the definition of those policies.

If the Thread Execution Scheduling option is defined, and the *schedpolicy* attribute specifies one of the priority-based policies defined under this option, the *schedparam* attribute contains the scheduling priority of the thread. A conforming implementation ensures that the priority value in *schedparam* is in the range associated with the scheduling policy when the thread attributes object is used to create a thread, or when the scheduling attributes of a thread are dynamically modified. The meaning of the priority value in *schedparam* is the same as that of *priority*.

TSP If _POSIX_THREAD_SPORADIC_SERVER is defined, the *schedparam* attribute supports four new members that are used for the sporadic server scheduling policy. These members are *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, and *sched_ss_max_repl*. The meaning of these attributes is the same as in the definitions that appear under Section 2.8.4 (on page 531).

When a process is created, its single thread has a scheduling policy and associated attributes equal to the policy and attributes of the process. The default scheduling contention scope value is implementation-defined. The default values of other scheduling attributes are implementation-defined.

19276 **Thread Scheduling Contention Scope**

19277 The scheduling contention scope of a thread defines the set of threads with which the thread
19278 competes for use of the processing resources. The scheduling operation selects at most one
19279 thread to execute on each processor at any point in time and the thread's scheduling attributes
19280 (for example, *priority*), whether under process scheduling contention scope or system scheduling
19281 contention scope, are the parameters used to determine the scheduling decision.

19282 The scheduling contention scope, in the context of scheduling a mixed scope environment,
19283 affects threads as follows:

19284 • A thread created with PTHREAD_SCOPE_SYSTEM scheduling contention scope contends
19285 for resources with all other threads in the same scheduling allocation domain relative to
19286 their system scheduling attributes. The system scheduling attributes of a thread created
19287 with PTHREAD_SCOPE_SYSTEM scheduling contention scope are the scheduling
19288 attributes with which the thread was created. The system scheduling attributes of a thread
19289 created with PTHREAD_SCOPE_PROCESS scheduling contention scope are the
19290 implementation-defined mapping into system attribute space of the scheduling attributes
19291 with which the thread was created.

19292 • Threads created with PTHREAD_SCOPE_PROCESS scheduling contention scope contend
19293 directly with other threads within their process that were created with
19294 PTHREAD_SCOPE_PROCESS scheduling contention scope. The contention is resolved
19295 based on the threads' scheduling attributes and policies. It is unspecified how such threads
19296 are scheduled relative to threads in other processes or threads with
19297 PTHREAD_SCOPE_SYSTEM scheduling contention scope.

19298 • Conforming implementations shall support the PTHREAD_SCOPE_PROCESS scheduling
19299 contention scope, the PTHREAD_SCOPE_SYSTEM scheduling contention scope, or both.

19300 **Scheduling Allocation Domain**

19301 Implementations shall support scheduling allocation domains containing one or more
19302 processors. It should be noted that the presence of multiple processors does not automatically
19303 indicate a scheduling allocation domain size greater than one. Conforming implementations on
19304 multi-processors may map all or any subset of the CPUs to one or multiple scheduling allocation
19305 domains, and could define these scheduling allocation domains on a per-thread, per-process, or
19306 per-system basis, depending on the types of applications intended to be supported by the
19307 implementation. The scheduling allocation domain is independent of scheduling contention
19308 scope, as the scheduling contention scope merely defines the set of threads with which a thread
19309 contends for processor resources, while scheduling allocation domain defines the set of
19310 processors for which it contends. The semantics of how this contention is resolved among
19311 threads for processors is determined by the scheduling policies of the threads.

19312 The choice of scheduling allocation domain size and the level of application control over
19313 scheduling allocation domains is implementation-defined. Conforming implementations may
19314 change the size of scheduling allocation domains and the binding of threads to scheduling
19315 allocation domains at any time.

19316 For application threads with scheduling allocation domains of size equal to one, the scheduling
19317 rules defined for SCHED_FIFO and SCHED_RR shall be used; see Scheduling Policies (on page
19318 531). All threads with system scheduling contention scope, regardless of the processes in which
19319 they reside, compete for the processor according to their priorities. Threads with process
19320 scheduling contention scope compete only with other threads with process scheduling
19321 contention scope within their process.

19322 For application threads with scheduling allocation domains of size greater than one, the rules

19323 TSP defined for SCHED_FIFO, SCHED_RR, and SCHED_SPORADIC shall be used in an
19324 implementation-defined manner. Each thread with system scheduling contention scope
19325 competes for the processors in its scheduling allocation domain in an implementation-defined
19326 manner according to its priority. Threads with process scheduling contention scope are
19327 scheduled relative to other threads within the same scheduling contention scope in the process.

19328 TSP If _POSIX_THREAD_SPORADIC_SERVER is defined, the rules defined for SCHED_SPORADIC
19329 in Scheduling Policies (on page 531) shall be used in an implementation-defined manner for
19330 application threads whose scheduling allocation domain size is greater than one.

19331 **Scheduling Documentation**

19332 If _POSIX_PRIORITY_SCHEDULING is defined, then any scheduling policies beyond
19333 TSP SCHED_OTHER, SCHED_FIFO, SCHED_RR, and SCHED_SPORADIC, as well as the effects of
19334 the scheduling policies indicated by these other values, and the attributes required in order to
19335 support such a policy, are implementation-defined. Furthermore, the implementation shall
19336 document the effect of all processor scheduling allocation domain values supported for these
19337 policies.

19338 2.9.5 Thread Cancellation

19339 The thread cancellation mechanism allows a thread to terminate the execution of any other
19340 thread in the process, except for threads created using *thrd_create*( ), in a controlled manner. The
19341 target thread (that is, the one that is being canceled) is allowed to hold cancellation requests
19342 pending in a number of ways and to perform application-specific cleanup processing when the
19343 notice of cancellation is acted upon.

19344 Cancellation is controlled by the cancellation control functions. Each thread maintains its own
19345 cancelability state. Cancellation may only occur at cancellation points or when the thread is
19346 asynchronously cancelable.

19347 The thread cancellation mechanism described in this section depends upon programs having set
19348 *deferred* cancelability state, which is specified as the default. Applications shall also carefully
19349 follow static lexical scoping rules in their execution behavior. For example, use of *setjmp*( ),
19350 *return*, *goto*, and so on, to leave user-defined cancellation scopes without doing the necessary
19351 scope pop operation results in undefined behavior.

19352 Use of asynchronous cancelability while holding resources which potentially need to be released
19353 may result in resource loss. Similarly, cancellation scopes may only be safely manipulated
19354 (pushed and popped) when the thread is in the *deferred* or *disabled* cancelability states.

19355 2.9.5.1 *Cancelability States*

19356 The cancelability state of a thread determines the action taken upon receipt of a cancellation
19357 request. The thread may control cancellation in a number of ways.

19358 Each thread maintains its own cancelability state, which may be encoded in two bits:

19359 1. Cancelability-Enable: When cancelability is PTHREAD_CANCEL_DISABLE (as defined
19360 in the Base Definitions volume of POSIX.1-2024, **<pthread.h>**), cancellation requests
19361 against the target thread are held pending. By default, cancelability is set to
19362 PTHREAD_CANCEL_ENABLE (as defined in **<pthread.h>**).

19363 2. Cancelability Type: When cancelability is enabled and the cancelability type is
19364 PTHREAD_CANCEL_ASYNCHRONOUS (as defined in **<pthread.h>**), new or pending
19365 cancellation requests may be acted upon at any time. When cancelability is enabled and

the cancelability type is PTHREAD_CANCEL_DEFERRED (as defined in **<pthread.h>**), cancellation requests are held pending until a cancellation point (see below) is reached. If cancelability is disabled, the setting of the cancelability type has no immediate effect as all cancellation requests are held pending; however, once cancelability is enabled again the new type is in effect. The cancelability type is PTHREAD_CANCEL_DEFERRED in all newly created threads including the thread in which *main*( ) was first invoked.

2.9.5.2 Cancellation Points

Cancellation points shall occur when a thread is executing the following functions:

accept( )	*msync*( )	*recvfrom*( )
accept4( )	*nanosleep*( )	*recvmsg*( )
aio_suspend( )	*open*( )	*select*( )
clock_nanosleep( )	*openat*( )	*send*( )
close( )	*pause*( )	*sendmsg*( )
cnd_timedwait( )	*poll*( )	*sendto*( )
cnd_wait( )	*posix_close*( )	*sigsuspend*( )
connect( )	*ppoll*( )	*sigtimedwait*( )
creat( )	*pread*( )	*sigwait*( )
fcntl( )†	*pselect*( )	*sigwaitinfo*( )
fdatasync( )	*pthread_cond_clockwait*( )	*sleep*( )
fsync( )	*pthread_cond_timedwait*( )	*tcdrain*( )
lockf( )††	*pthread_cond_wait*( )	*thrd_join*( )
mq_receive( )	*pthread_join*( )	*thrd_sleep*( )
mq_send( )	*pthread_testcancel*( )	*wait*( )
mq_timedreceive( )	*pwrite*( )	*waitid*( )
mq_timedsend( )	*read*( )	*waitpid*( )
msgrcv( )	*readv*( )	*write*( )
msgsnd( )	*recv*( )	*writev*( )

A cancellation point may also occur when a thread is executing the following functions:

---

† When the *cmd* argument is F_SETLKW or F_OFD_SETLKW.

†† When the *function* argument is F_LOCK.

19396 *access*( )
19397 *bindtextdomain*( )
19398 *catclose*( )
19399 *catopen*( )
19400 *chmod*( )
19401 *chown*( )
19402 *closedir*( )
19403 *closelog*( )
19404 *ctermid*( )
19405 *dcgettext*( )
19406 *dcgettext_l*( )
19407 *dcngettext*( )
19408 *dcngettext_l*( )
19409 *dgettext*( )
19410 *dgettext_l*( )
19411 *dlclose*( )
19412 *dlopen*( )
19413 *dngettext*( )
19414 *dngettext_l*( )
19415 *dprintf*( )
19416 *endhostent*( )
19417 *endnetent*( )
19418 *endprotoent*( )
endservent( )
faccessat( )
fchmod( )
fchmodat( )
fchown( )
fchownat( )
fclose( )
fcntl( )†††
fflush( )
fgetc( )
fgetpos( )
fgets( )
fgetwc( )
fgetws( )
fmtmsg( )
fopen( )
fpathconf( )
fprintf( )
fputc( )
fputs( )
fputwc( )
fputws( )
fread( )
freopen( )
fscanf( )
fseek( )
fseeko( )
fsetpos( )
fstat( )
fstatat( )
ftell( )
ftello( )
futimens( )
fwprintf( )
fwrite( )
fwscanf( )
getaddrinfo( )
getc( )
getc_unlocked( )
getchar( )
getchar_unlocked( )
getcwd( )
getdelim( )
getgrgid_r( )
getgrnam_r( )
gethostid( )

19419 ††† For any value of the *cmd* argument.

19420 *gethostname*( )
19421 *getline*( )
19422 *getlogin_r*( )
19423 *getnameinfo*( )
19424 *getpwnam_r*( )
19425 *getpwuid_r*( )
19426 *gettext*( )
19427 *gettext_l*( )
19428 *getwc*( )
19429 *getwchar*( )
19430 *glob*( )
19431 *iconv_close*( )
19432 *iconv_open*( )
19433 *link*( )
19434 *linkat*( )
19435 *lio_listio*( )
19436 *localtime_r*( )
19437 *lockf*( )
19438 *lseek*( )
19439 *lstat*( )
19440 *mkdir*( )
19441 *mkdirat*( )
19442 *mkdtemp*( )
19443 *mkfifo*( )
19444 *mkfifoat*( )
19445 *mknod*( )
19446 *mknodat*( )
19447 *mkstemp*( )
19448 *mktime*( )
19449 *ngettext*( )
19450 *ngettext_l*( )
19451 *opendir*( )
19452 *openlog*( )
19453 *pathconf*( )
19454 *perror*( )
19455 *popen*( )
19456 *posix_devctl*( )
posix_fadvise( )
posix_fallocate( )
posix_getdents( )
posix_madvise( )
posix_openpt( )
posix_spawn( )
posix_spawnp( )
posix_typed_mem_open( )
printf( )
psiginfo( )
psignal( )
pthread_rwlock_clockrdlock( )
pthread_rwlock_clockwrlock( )
pthread_rwlock_rdlock( )
pthread_rwlock_timedrdlock( )
pthread_rwlock_timedwrlock( )
pthread_rwlock_wrlock( )
ptsname( )
ptsname_r( )
putc( )
putc_unlocked( )
putchar( )
putchar_unlocked( )
puts( )
putwc( )
putwchar( )
readdir_r( )
readlink( )
readlinkat( )
remove( )
rename( )
renameat( )
rewind( )
rewinddir( )
scandir( )
scanf( )
seekdir( )
sem_clockwait( )
sem_timedwait( )
sem_wait( )
semop( )
sethostent( )
setnetent( )
setprotoent( )
setservent( )
stat( )
strerror_l( )
strerror_r( )
strftime( )
strftime_l( )
symlink( )
symlinkat( )
sync( )
syslog( )
tmpfile( )
tmpnam( )
ttyname_r( )
tzset( )
ungetc( )
ungetwc( )
unlink( )
unlinkat( )
utimensat( )
utimes( )
vdprintf( )
vfprintf( )
vfwprintf( )
vprintf( )
vwprintf( )
wcsftime( )
wordexp( )
wprintf( )
wscanf( )

19457 In addition, a cancellation point may occur when a thread is executing any function that this
19458 standard does not require to be thread-safe but the implementation documents as being thread-
19459 safe. If a thread is cancelled while executing a non-thread-safe function, the behavior is
19460 undefined.

19461 An implementation shall not introduce cancellation points into any other functions specified in
19462 this volume of POSIX.1-2024.

19463 The side-effects of acting upon a cancellation request while suspended during a call of a function
19464 are the same as the side-effects that may be seen in a single-threaded program when a call to a
19465 function is interrupted by a signal and the given function returns [EINTR]. Any such side-
19466 effects occur before any cancellation cleanup handlers are called. For functions that are explicitly
19467 required not to return when interrupted (for example, *pclose*( )), if a thread is canceled while
19468 executing the function, the behavior is undefined.

19469 Whenever a thread has cancelability enabled and a cancellation request has been made with that
19470 thread as the target, and the thread then calls any function that is a cancellation point (such as

pthread_testcancel( ) or *read*( )), the cancellation request shall be acted upon before the function returns. If a thread has cancelability enabled and a cancellation request is made with the thread as a target while the thread is suspended at a cancellation point, the thread shall be awakened and the cancellation request shall be acted upon. It is unspecified whether the cancellation request is acted upon or whether the cancellation request remains pending and the thread resumes normal execution if the thread is suspended at a cancellation point and either:

- The event for which it is waiting occurs
- A specified timeout expires

before the cancellation request is acted upon.

2.9.5.3 Thread Cancellation Cleanup Handlers

Each thread that was not created using *thrd_create*( ) maintains a list of cancellation cleanup handlers. The programmer uses the *pthread_cleanup_push*( ) and *pthread_cleanup_pop*( ) functions to place routines on and remove routines from this list.

When a cancellation request is acted upon, or when a thread calls *pthread_exit*( ), the thread first disables cancellation by setting its cancelability state to PTHREAD_CANCEL_DISABLE and its cancelability type to PTHREAD_CANCEL_DEFERRED. The cancelability state shall remain set to PTHREAD_CANCEL_DISABLE until the thread has terminated. The behavior is undefined if a cancellation cleanup handler or thread-specific data destructor routine changes the cancelability state to PTHREAD_CANCEL_ENABLE.

The routines in the thread's list of cancellation cleanup handlers shall be invoked one by one in LIFO sequence; that is, the last routine pushed onto the list (Last In) is the first to be invoked (First Out). When the cancellation cleanup handler for a scope is invoked, the storage for that scope remains valid. If the last cancellation cleanup handler returns, thread-specific data destructors (if any) associated with thread-specific data keys for which the thread has non-NULL values shall be run, in unspecified order, as described for *pthread_key_create*( ) and *tss_create*( ).

After all cancellation cleanup handlers and thread-specific data destructors have returned, thread execution is terminated. If the thread has terminated because of a call to *pthread_exit*( ), the *value_ptr* argument is made available to any threads joining with the target. If the thread has terminated by acting on a cancellation request, a status of PTHREAD_CANCELED is made available to any threads joining with the target. The symbolic constant PTHREAD_CANCELED expands to a constant expression of type (**void** *) whose value matches no pointer to an object in memory nor the value NULL.

A side-effect of acting upon a cancellation request while in a condition variable wait is that the mutex is re-acquired before calling the first cancellation cleanup handler. In addition, the thread is no longer considered to be waiting for the condition and the thread shall not have consumed any pending condition signals on the condition.

A cancellation cleanup handler cannot exit via *longjmp*( ) or *siglongjmp*( ).

2.9.5.4 Async-Cancel Safety

The *pthread_cancel*( ), *pthread_setcancelstate*( ), and *pthread_setcanceltype*( ) functions are defined to be async-cancel safe.

No other functions in this volume of POSIX.1-2024 are required to be async-cancel-safe.

If a thread has asynchronous cancellation enabled and is cancelled during execution of a

function that is not async-cancel-safe, the behavior is undefined.

If a thread has deferred cancellation enabled, a signal-catching function is called in that thread during execution of a function that is not async-cancel-safe, and the signal-catching function calls any function that is a cancellation point while a cancellation is pending for the thread, without first disabling cancellation, the behavior is undefined.

2.9.6 Thread Read-Write Locks

Multiple readers, single writer (read-write) locks allow many threads to have simultaneous read-only access to data while allowing only one thread to have exclusive write access at any given time. They are typically used to protect data that is read more frequently than it is changed.

One or more readers acquire read access to the resource by performing a read lock operation on the associated read-write lock. A writer acquires exclusive write access by performing a write lock operation. Basically, all readers exclude any writers and a writer excludes all readers and any other writers.

A thread that has blocked on a read-write lock (for example, has not yet returned from a *pthread_rwlock_rdlock*( ) or *pthread_rwlock_wrlock*( ) call) shall not prevent any unblocked thread that is eligible to use the same processing resources from eventually making forward progress in its execution. Eligibility for processing resources shall be determined by the scheduling policy.

Read-write locks can be used to synchronize threads in the current process and other processes if they are allocated in memory that is writable and shared among the cooperating processes and have been initialized for this behavior.

2.9.7 Thread Interactions with File Operations

All of the following functions shall be atomic with respect to each other in the effects specified in POSIX.1-2024 when they operate on files in the file hierarchy:

chmod( )	*fchownat*( )	*link*( )	*readlinkat*( )	*truncate*( )
chown( )	*fstat*( )	*linkat*( )	*rename*( )	*unlink*( )
creat( )	*fstatat*( )	*lstat*( )	*renameat*( )	*unlinkat*( )
fchmod( )	*ftruncate*( )	*open*( )	*stat*( )	*utimensat*( )
fchmodat( )	*futimens*( )	*openat*( )	*symlink*( )	*utimes*( )
fchown( )	*lchown*( )	*readlink*( )	*symlinkat*( )	

If two threads each call one of these functions, each call shall either see all of the specified effects of the other call, or none of them.

Except where specified otherwise, all of the following functions shall be atomic with respect to each other in the effects specified in POSIX.1-2024 when they operate on file descriptors that are open, or being opened, to files in the file hierarchy:

close( )	*fstat*( )	*lseek*( )	*read*( )	*writev*( )
dup2( )	*fstatat*( )	*open*( )	*readv*( )	
dup3( )	*ftruncate*( )	*openat*( )	*pwrite*( )	
fcntl( )	*futimens*( )	*pread*( )	*write*( )	

If two threads each call one of these functions, each call shall either see all of the specified effects of the other call, or none of them. The requirement on the *close*( ) function shall also apply whenever a file descriptor is successfully closed, however caused (for example, as a consequence

of calling *close*( ), calling *dup2*( ), or of process termination).

2.9.8 Use of Application-Managed Thread Stacks

An ``application-managed thread stack'' is a region of memory allocated by the application—for example, memory returned by the *malloc*( ) or *mmap*( ) functions—and designated as a stack through the act of passing the address and size of the stack, respectively, as the *stackaddr* and *stacksize* arguments to *pthread_attr_setstack*( ). Application-managed stacks allow the application to precisely control the placement and size of a stack.

The application grants to the implementation permanent ownership of and control over the application-managed stack when the attributes object in which the *stack* or *stackaddr* attribute has been set is used, either by presenting that attribute's object as the *attr* argument in a call to *pthread_create*( ) that completes successfully, or by storing a pointer to the attributes object in the *sigev_notify_attributes* member of a **struct sigevent** and passing that **struct sigevent** to a function accepting such argument that completes successfully. The application may thereafter utilize the memory within the stack only within the normal context of stack usage within or properly synchronized with a thread that has been scheduled by the implementation with stack pointer value(s) that are within the range of that stack. In particular, the region of memory cannot be freed, nor can it be later specified as the stack for another thread.

When specifying an attributes object with an application-managed stack through the *sigev_notify_attributes* member of a **struct sigevent**, the results are undefined if the requested signal is generated multiple times (as for a repeating timer).

Until an attributes object in which the *stack* or *stackaddr* attribute has been set is used, the application retains ownership of and control over the memory allocated to the stack. It may free or reuse the memory as long as it either deletes the attributes object, or before using the attributes object replaces the stack by making an additional call to *pthread_attr_setstack*( ), that was used originally to designate the stack. There is no mechanism to retract the reference to an application-managed stack by an existing attributes object.

Once an attributes object with an application-managed stack has been used, that attributes object cannot be used again by a subsequent call to *pthread_create*( ) or any function accepting a **struct sigevent** with *sigev_notify_attributes* containing a pointer to the attributes object, without designating an unused application-managed stack by making an additional call to *pthread_attr_setstack*( ).

2.9.9 Synchronization Object Copies and Alternative Mappings

TSH For barriers, condition variables, mutexes, and read-write locks, if the process-shared attribute is set to PTHREAD_PROCESS_PRIVATE, only the synchronization object at the address used to initialize it can be used for performing synchronization. The effect of referring to another TSH mapping of the same object when locking, unlocking, or destroying the object is undefined. If the process-shared attribute is set to PTHREAD_PROCESS_SHARED, only the synchronization object itself can be used for performing synchronization; however, it need not be referenced at the address used to initalize it (that is, another mapping of the same object can be used). The effect of referring to a copy of the object when locking, unlocking, or destroying it is undefined.

For spin locks, the above requirements shall apply as if spin locks have a process-shared attribute that is set from the *pshared* argument to *pthread_spin_init*( ). For semaphores, the above requirements shall apply as if semaphores have a process-shared attribute that is set to PTHREAD_PROCESS_PRIVATE if the *pshared* argument to *sem_init*( ) is zero and set to PTHREAD_PROCESS_SHARED if *pshared* is non-zero.

For ISO C functions declared in **<threads.h>**, the above requirements shall apply as if condition variables of type **cnd_t** and mutexes of type **mtx_t** have a process-shared attribute that is set to PTHREAD_PROCESS_PRIVATE.

2.10 Sockets

A socket is an endpoint for communication using the facilities described in this section. A socket is created with a specific socket type, described in Section 2.10.6 (on page 550), and is associated with a specific protocol, detailed in Section 2.10.3. A socket is accessed via a file descriptor obtained when the socket is created.

2.10.1 Address Families

All network protocols are associated with a specific address family. An address family provides basic services to the protocol implementation to allow it to function within a specific network environment. These services may include packet fragmentation and reassembly, routing, addressing, and basic transport. An address family is normally comprised of a number of protocols, one per socket type. Each protocol is characterized by an abstract socket type. It is not required that an address family support all socket types. An address family may contain multiple protocols supporting the same socket abstraction.

Section 2.10.17 (on page 557), Section 2.10.19 (on page 558), and Section 2.10.20 (on page 558), respectively, describe the use of sockets for local UNIX connections, for Internet protocols based on IPv4, and for Internet protocols based on IPv6.

2.10.2 Addressing

An address family defines the format of a socket address. All network addresses are described using a general structure, called a **sockaddr**, as defined in the Base Definitions volume of POSIX.1-2024, **<sys/socket.h>**. However, each address family imposes finer and more specific structure, generally defining a structure with fields specific to the address family. The field *sa_family* in the **sockaddr** structure contains the address family identifier, specifying the format of the *sa_data* area. The size of the *sa_data* area is unspecified.

2.10.3 Protocols

A protocol supports one of the socket abstractions detailed in Section 2.10.6 (on page 550). Selecting a protocol involves specifying the address family, socket type, and protocol number to the *socket*( ) function. Certain semantics of the basic socket abstractions are protocol-specific. All protocols are expected to support the basic model for their particular socket type, but may, in addition, provide non-standard facilities or extensions to a mechanism.

2.10.4 Routing

Sockets provides packet routing facilities. A routing information database is maintained, which is used in selecting the appropriate network interface when transmitting packets.

2.10.5 Interfaces

Each network interface in a system corresponds to a path through which messages can be sent and received. A network interface usually has a hardware device associated with it, though certain interfaces such as the loopback interface, do not.

2.10.6 Socket Types

A socket is created with a specific type, which defines the communication semantics and which allows the selection of an appropriate communication protocol. Four types are defined: RS SOCK_DGRAM, SOCK_RAW, SOCK_SEQPACKET, and SOCK_STREAM. Implementations may specify additional socket types.

The SOCK_STREAM socket type provides reliable, sequenced, full-duplex octet streams between the socket and a peer to which the socket is connected. A socket of type SOCK_STREAM needs to be in a connected state before any data can be sent or received. Record boundaries are not maintained; data sent on a stream socket using output operations of one size can be received using input operations of smaller or larger sizes without loss of data. Data may be buffered; successful return from an output function does not imply that the data has been delivered to the peer or even transmitted from the local system. If data cannot be successfully transmitted within a given time then the connection is considered broken, and subsequent operations shall fail. A SIGPIPE signal is raised if a thread attempts to send data on a broken stream (one that is no longer connected), except that the signal is suppressed if the MSG_NOSIGNAL flag is used in calls to *send*( ), *sendto*( ), and *sendmsg*( ). Support for an out-of-band data transmission facility is protocol-specific.

The SOCK_SEQPACKET socket type is similar to the SOCK_STREAM type, and is also connection-oriented. The only difference between these types is that record boundaries are maintained using the SOCK_SEQPACKET type. A record can be sent using one or more output operations and received using one or more input operations, but a single operation never transfers parts of more than one record. Record boundaries are visible to the receiver via the MSG_EOR flag in the received message flags returned by the *recvmsg*( ) function. It is protocol-specific whether a maximum record size is imposed.

The SOCK_DGRAM socket type supports connectionless data transfer which is not necessarily acknowledged or reliable. Datagrams can be sent to the address specified (possibly multicast or broadcast) in each output operation, and incoming datagrams can be received from multiple sources. The source address of each datagram is available when receiving the datagram. An application can also pre-specify a peer address, in which case calls to output functions that do not specify a peer address shall send to the pre-specified peer. If a peer has been specified, only datagrams from that peer shall be received. A datagram shall be sent in a single output operation, and needs to be received in a single input operation. The maximum size of a datagram is protocol-specific; with some protocols, the limit is implementation-defined. Output datagrams may be buffered within the system; thus, a successful return from an output function does not guarantee that a datagram is actually sent or received. However, implementations should attempt to detect any errors possible before the return of an output function, reporting any error by an unsuccessful return value.

19677 RS The SOCK_RAW socket type is similar to the SOCK_DGRAM type. It differs in that it is
19678 normally used with communication providers that underlie those used for the other socket
19679 types. For this reason, the creation of a socket with type SOCK_RAW shall require appropriate
19680 privileges. The format of datagrams sent and received with this socket type generally include
19681 specific protocol headers, and the formats are protocol-specific and implementation-defined.

19682 2.10.7 Socket I/O Mode

19683 The I/O mode of a socket is described by the O_NONBLOCK file status flag which pertains to
19684 the open file description for the socket. This flag is initially off when a socket is created, but may
19685 be set and cleared by the use of the F_SETFL command of the *fcntl*( ) function.

19686 When the O_NONBLOCK flag is set, certain functions that would normally block until they are
19687 complete shall return immediately.

19688 The *bind*( ) function initiates an address assignment and shall return without blocking when
19689 O_NONBLOCK is set; if the socket address cannot be assigned immediately, *bind*( ) shall return
19690 the [EINPROGRESS] error to indicate that the assignment was initiated successfully, but that it
19691 has not yet completed.

19692 The *connect*( ) function initiates a connection and shall return without blocking when
19693 O_NONBLOCK is set; it shall return the error [EINPROGRESS] to indicate that the connection
19694 was initiated successfully, but that it has not yet completed.

19695 Data transfer operations (the *read*( ), *write*( ), *send*( ), and *recv*( ) functions) shall complete
19696 immediately, transfer only as much as is available, and then return without blocking, or return
19697 an error indicating that no transfer could be made without blocking.

19698 2.10.8 Socket Owner

19699 The owner of a socket is unset when a socket is created. The owner may be set to a process ID or
19700 process group ID using the F_SETOWN command of the *fcntl*( ) function.

19701 2.10.9 Socket Queue Limits

19702 The transmit and receive queue sizes for a socket are set when the socket is created. The default
19703 sizes used are both protocol-specific and implementation-defined. The sizes may be changed
19704 using the *setsockopt*( ) function.

19705 2.10.10 Pending Error

19706 Errors may occur asynchronously, and be reported to the socket in response to input from the
19707 network protocol. The socket stores the pending error to be reported to a user of the socket at the
19708 next opportunity. The error is returned in response to a subsequent *send*( ), *recv*( ), or *getsockopt*( )
19709 operation on the socket, and the pending error is then cleared.

2.10.11 Socket Receive Queue

A socket has a receive queue that buffers data when it is received by the system until it is removed by a receive call. Depending on the type of the socket and the communication provider, the receive queue may also contain ancillary data such as the addressing and other protocol data associated with the normal data in the queue, and may contain out-of-band or expedited data. The limit on the queue size includes any normal, out-of-band data, datagram source addresses, and ancillary data in the queue. The description in this section applies to all sockets, even though some elements cannot be present in some instances.

The contents of a receive buffer are logically structured as a series of data segments with associated ancillary data and other information. A data segment may contain normal data or out-of-band data, but never both. A data segment may complete a record if the protocol supports records (always true for types SOCK_SEQPACKET and SOCK_DGRAM). A record may be stored as more than one segment; the complete record might never be present in the receive buffer at one time, as a portion might already have been returned to the application, and another portion might not yet have been received from the communications provider. A data segment may contain ancillary protocol data, which is logically associated with the segment. Ancillary data is received as if it were queued along with the first normal data octet in the segment (if any). A segment may contain ancillary data only, with no normal or out-of-band data. For the purposes of this section, a datagram is considered to be a data segment that terminates a record, and that includes a source address as a special type of ancillary data. Data segments are placed into the queue as data is delivered to the socket by the protocol. Normal data segments are placed at the end of the queue as they are delivered. If a new segment contains the same type of data as the preceding segment and includes no ancillary data, and if the preceding segment does not terminate a record, the segments are logically merged into a single segment.

The receive queue is logically terminated if an end-of-file indication has been received or a connection has been terminated. A segment shall be considered to be terminated if another segment follows it in the queue, if the segment completes a record, or if an end-of-file or other connection termination has been reported. The last segment in the receive queue shall also be considered to be terminated while the socket has a pending error to be reported.

A receive operation shall never return data or ancillary data from more than one segment.

2.10.12 Socket Out-of-Band Data State

The handling of received out-of-band data is protocol-specific. Out-of-band data may be placed in the socket receive queue, either at the end of the queue or before all normal data in the queue. In this case, out-of-band data is returned to an application program by a normal receive call. Out-of-band data may also be queued separately rather than being placed in the socket receive queue, in which case it shall be returned only in response to a receive call that requests out-of-band data. It is protocol-specific whether an out-of-band data mark is placed in the receive queue to demarcate data preceding the out-of-band data and following the out-of-band data. An out-of-band data mark is logically an empty data segment that cannot be merged with other segments in the queue. An out-of-band data mark is never returned in response to an input operation. The *sockatmark*( ) function can be used to test whether an out-of-band data mark is the first element in the queue. If an out-of-band data mark is the first element in the queue when an input function is called without the MSG_PEEK option, the mark is removed from the queue and the following data (if any) is processed as if the mark had not been present.

2.10.13 Connection Indication Queue

Sockets that are used to accept incoming connections maintain a queue of outstanding connection indications. This queue is a list of connections that are awaiting acceptance by the application; see *listen*( ).

2.10.14 Signals

One category of event at the socket interface is the generation of signals. These signals report protocol events or process errors relating to the state of the socket. The generation or delivery of a signal does not change the state of the socket, although the generation of the signal may have been caused by a state change.

The SIGPIPE signal shall be sent to a thread that attempts to send data on a socket that is no longer able to send (one that is no longer connected), except that the signal is suppressed if the MSG_NOSIGNAL flag is used in calls to *send*( ), *sendto*( ), and *sendmsg*( ). Regardless of whether the generation of the signal is suppressed, the send operation shall fail with the [EPIPE] error.

If a socket has an owner, the SIGURG signal is sent to the owner of the socket when it is notified of expedited or out-of-band data. The socket state at this time is protocol-dependent, and the status of the socket is specified in Section 2.10.17 (on page 557), Section 2.10.19 (on page 558), and Section 2.10.20 (on page 558). Depending on the protocol, the expedited data may or may not have arrived at the time of signal generation.

2.10.15 Asynchronous Errors

If any of the following conditions occur asynchronously for a socket, the corresponding value listed below shall become the pending error for the socket:

[ECONNABORTED]
The connection was aborted locally.

[ECONNREFUSED]
For a connection-mode socket attempting a non-blocking connection, the attempt to connect was forcefully rejected. For a connectionless-mode socket, an attempt to deliver a datagram was forcefully rejected.

[ECONNRESET]
The peer has aborted the connection.

[EHOSTUNREACH]
The destination host is not reachable.

[EMSGSIZE]
For a connectionless-mode socket, the size of a previously sent datagram prevented delivery.

[ENETDOWN]
The local network connection is not operational.

[ENETRESET]
The connection was aborted by the network.

[ENETUNREACH]
The destination network is not reachable.

2.10.16 Use of Options

There are a number of socket options which either specialize the behavior of a socket or provide useful information. These options may be set at different protocol levels and are always present at the uppermost ``socket'' level.

Socket options are manipulated by two functions, *getsockopt*( ) and *setsockopt*( ). These functions allow an application program to customize the behavior and characteristics of a socket to provide the desired effect.

All of the options usable with *setsockopt*( ) have defaults. For each option where a default value is listed as implementation-defined, the implementation also controls whether a socket created by *accept*( ) or *accept4*( ) starts with the option reset to the original default value, or inherited as the value previously customized on the original listening socket. The type and meaning of these values is defined by the protocol level to which they apply. Instead of using the default values, an application program may choose to customize one or more of the options. However, in the bulk of cases, the default values are sufficient for the application.

Some of the options are used to enable or disable certain behavior within the protocol modules (for example, turn on debugging) while others may be used to set protocol-specific information (for example, IP time-to-live on all the application's outgoing packets). As each of the options is introduced, its effect on the underlying protocol modules is described.

Table 2-1 shows the value for the socket level.

Table 2-1 Value of Level for Socket Options

Name	Description
SOL_SOCKET	Options are intended for the sockets level.

Table 2-2 (on page 555) lists those options present at the socket level; that is, when the *level* parameter of the *getsockopt*( ) or *setsockopt*( ) function is SOL_SOCKET, the types of the option value parameters associated with each option, and a brief synopsis of the meaning of the option value parameter. Unless otherwise noted, each may be examined with *getsockopt*( ) and set with *setsockopt*( ) on all types of socket. Options at other protocol levels vary in format and name.

Table 2-2 Socket-Level Options

Option	Parameter Type	Parameter Meaning
SO_ACCEPTCONN	**int**	Non-zero indicates that socket listening is enabled (*getsockopt*( ) only).
SO_BROADCAST	**int**	Non-zero requests permission to transmit broadcast datagrams (SOCK_DGRAM sockets only).
SO_DEBUG	**int**	Non-zero requests debugging in underlying protocol modules.
SO_DOMAIN	**int**	Identify socket domain (*getsockopt*( ) only).
SO_DONTROUTE	**int**	Non-zero requests bypass of normal routing; route based on destination address only.
SO_ERROR	**int**	Requests and clears pending error information on the socket (*getsockopt*( ) only).
SO_KEEPALIVE	**int**	Non-zero requests periodic transmission of keepalive messages (protocol-specific).
SO_LINGER	**struct linger**	Specify actions to be taken for queued, unsent data on *close*( ): linger on/off and linger time in seconds.
SO_OOBINLINE	**int**	Non-zero requests that out-of-band data be placed into normal data input queue as received.
SO_PROTOCOL	**int**	Identify socket protocol (*getsockopt*( ) only).
SO_RCVBUF	**int**	Size of receive buffer (in bytes).
SO_RCVLOWAT	**int**	Minimum amount of data to return to application for input operations (in bytes).
SO_RCVTIMEO	**struct timeval**	Timeout value for a socket receive operation.
SO_REUSEADDR	**int**	Non-zero requests reuse of local addresses in *bind*( ) (protocol-specific).
SO_SNDBUF	**int**	Size of send buffer (in bytes).
SO_SNDLOWAT	**int**	Minimum amount of data to send for output operations (in bytes).
SO_SNDTIMEO	**struct timeval**	Timeout value for a socket send operation.
SO_TYPE	**int**	Identify socket type (*getsockopt*( ) only).

The SO_ACCEPTCONN option is used only on *getsockopt*( ). When this option is specified, *getsockopt*( ) shall report whether socket listening is enabled for the socket. A value of zero shall indicate that socket listening is disabled; non-zero that it is enabled. SO_ACCEPTCONN has no default value.

The SO_BROADCAST option requests permission to send broadcast datagrams on the socket. Support for SO_BROADCAST is protocol-specific. The default for SO_BROADCAST is that the ability to send broadcast datagrams on a socket is disabled.

The SO_DEBUG option enables debugging in the underlying protocol modules. This can be useful for tracing the behavior of the underlying protocol modules during normal system operation. The semantics of the debug reports are implementation-defined. The default value for SO_DEBUG is for debugging to be turned off.

The SO_DOMAIN option is used only on *getsockopt*( ). When this option is specified, *getsockopt*( ) shall return the domain of the socket (for example, AF_INET6). SO_DOMAIN has no default value.

The SO_DONTROUTE option requests that outgoing messages bypass the standard routing

facilities. The destination needs to be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. It is protocol-specific whether this option has any effect and how the outgoing network interface is chosen. Support for this option with each protocol is implementation-defined.

The SO_ERROR option is used only on *getsockopt*(). When this option is specified, *getsockopt*() shall return any pending error on the socket and clear the error status. It shall return a value of 0 if there is no pending error. SO_ERROR may be used to check for asynchronous errors on connected connectionless-mode sockets or for other types of asynchronous errors. SO_ERROR has no default value.

The SO_KEEPALIVE option enables the periodic transmission of messages on a connected socket. The behavior of this option is protocol-specific. On a connection-mode socket for which a connection has been established, if SO_KEEPALIVE is enabled and the connected socket fails to respond to the keep-alive messages, the connection shall be broken. The default value for SO_KEEPALIVE is zero, specifying that this capability is turned off.

The SO_LINGER option controls the action of the interface when unsent messages are queued on a socket and a *close*() is performed. The details of this option are protocol-specific. If SO_LINGER is enabled, the system shall block the calling thread during *close*() until it can transmit the data or until the end of the interval indicated by the *l_linger* member, whichever comes first. If SO_LINGER is not specified, and *close*() is issued, the system handles the call in a way that allows the calling thread to continue as quickly as possible. The default value for SO_LINGER is zero, or off, for the *l_onoff* element of the option value and zero seconds for the linger time specified by the *l_linger* element.

The SO_OOBINLINE option is valid only on protocols that support out-of-band data. The SO_OOBINLINE option requests that out-of-band data be placed in the normal data input queue as received; it is then accessible using the *read*() or *recv*() functions without the MSG_OOB flag set. The default for SO_OOBINLINE is off; that is, for out-of-band data not to be placed in the normal data input queue.

The SO_PROTOCOL option is used only on *getsockopt*(). When this option is specified, *getsockopt*() shall return the socket protocol (for example, IPPROTO_TCP). SO_PROTOCOL has no default value.

The SO_RCVBUF option requests that the buffer space allocated for receive operations on this socket be set to the value, in bytes, of the option value. Applications may wish to increase buffer size for high volume connections, or may decrease buffer size to limit the possible backlog of incoming data. The default value for the SO_RCVBUF option value is implementation-defined, and may vary by protocol.

The SO_RCVLOWAT option sets the minimum number of bytes to process for socket input operations. In general, receive calls block until any (non-zero) amount of data is received, then return the smaller of the amount available or the amount requested. The default value for SO_RCVLOWAT is 1, and does not affect the general case. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. Receive calls may still return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different from that returned (for example, out-of-band data). As mentioned previously, the default value for SO_RCVLOWAT is 1 byte. It is implementation-defined whether the SO_RCVLOWAT option can be set.

The SO_RCVTIMEO option is an option to set a timeout value for input operations. It accepts a **timeval** structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without receiving additional data, it shall return with a partial count or *errno* shall be set to

[EAGAIN] or [EWOULDBLOCK] if no data were received. The default for this option is the value zero, which indicates that a receive operation will not time out. It is implementation-defined whether the SO_RCVTIMEO option can be set.

The SO_REUSEADDR option indicates that the rules used in validating addresses supplied in a *bind*( ) should allow reuse of local addresses. Operation of this option is protocol-specific. The default value for SO_REUSEADDR is off; that is, reuse of local addresses is not permitted.

The SO_SNDBUF option requests that the buffer space allocated for send operations on this socket be set to the value, in bytes, of the option value. The default value for the SO_SNDBUF option value is implementation-defined, and may vary by protocol.

The SO_SNDLOWAT option sets the minimum number of bytes to process for socket output operations. Most output operations process all of the data supplied by the call, delivering data to the protocol for transmission and blocking as necessary for flow control. Non-blocking output operations process as much data as permitted subject to flow control without blocking, but process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. A *select*( ) operation testing the ability to write to a socket shall return true only if the send low water mark could be processed. The default value for SO_SNDLOWAT is implementation-defined and protocol-specific. It is implementation-defined whether the SO_SNDLOWAT option can be set.

The SO_SNDTIMEO option is an option to set a timeout value for the amount of time that an output function shall block because flow control prevents data from being sent. As noted in Table 2-2 (on page 555), the option value is a **timeval** structure with the number of seconds and microseconds specifying the limit on how long to wait for an output operation to complete. If a send operation has blocked for this much time, it shall return with a partial count or *errno* set to [EAGAIN] or [EWOULDBLOCK] if no data were sent. The default for this option is the value zero, which indicates that a send operation will not time out. It is implementation-defined whether the SO_SNDTIMEO option can be set.

The SO_TYPE option is used only on *getsockopt*( ). When this option is specified, *getsockopt*( ) shall return the type of the socket (for example, SOCK_STREAM). This option is useful to servers that inherit sockets on start-up. SO_TYPE has no default value.

2.10.17 Use of Sockets for Local UNIX Connections

Support for UNIX domain sockets is mandatory.

UNIX domain sockets provide process-to-process communication in a single system.

2.10.17.1 Headers

The symbolic constant AF_UNIX defined in the **<sys/socket.h>** header is used to identify the UNIX domain address family. The **<sys/un.h>** header contains other definitions used in connection with UNIX domain sockets. See XBD Chapter 14 (on page 221).

The **sockaddr_storage** structure defined in **<sys/socket.h>** shall be large enough to accommodate a **sockaddr_un** structure (see the **<sys/un.h>** header defined in XBD Chapter 14, on page 221) and shall be aligned at an appropriate boundary so that pointers to it can be cast as pointers to **sockaddr_un** structures and used to access the fields of those structures without alignment problems. When a **sockaddr_storage** structure is cast as a **sockaddr_un** structure, the *ss_family* field maps onto the *sun_family* field.

2.10.18 Use of Sockets over Internet Protocols

When a socket is created in the Internet family with a protocol value of zero, the implementation shall use the protocol listed below for the type of socket created.

SOCK_STREAM	IPPROTO_TCP.
SOCK_DGRAM	IPPROTO_UDP.
RS SOCK_RAW	IPPROTO_RAW.
SOCK_SEQPACKET	Unspecified.

RS A raw interface to IP is available by creating an Internet socket of type SOCK_RAW. The default protocol for type SOCK_RAW shall be identified in the IP header with the value IPPROTO_RAW. Applications should not use the default protocol when creating a socket with type SOCK_RAW, but should identify a specific protocol by value. The ICMP control protocol is accessible from a raw socket by specifying a value of IPPROTO_ICMP for protocol.

2.10.19 Use of Sockets over Internet Protocols Based on IPv4

Support for sockets over Internet protocols based on IPv4 is mandatory. IPv4 is described in RFC 791.

2.10.19.1 Headers

The symbolic constant AF_INET defined in the **<sys/socket.h>** header is used to identify the IPv4 Internet address family. The **<netinet/in.h>** header contains other definitions used in connection with IPv4 Internet sockets. See XBD Chapter 14 (on page 221).

The **sockaddr_storage** structure defined in **<sys/socket.h>** shall be large enough to accommodate a **sockaddr_in** structure (see the **<netinet/in.h>** header defined in XBD Chapter 14, on page 221) and shall be aligned at an appropriate boundary so that pointers to it can be cast as pointers to **sockaddr_in** structures and used to access the fields of those structures without alignment problems. When a **sockaddr_storage** structure is cast as a **sockaddr_in** structure, the *ss_family* field maps onto the *sin_family* field.

2.10.20 Use of Sockets over Internet Protocols Based on IPv6

IP6 This section describes extensions to support sockets over Internet protocols based on IPv6. The functionality described in this section shall be provided on implementations that support the IPV6 option (and the rest of this section is not further shaded for this option).

IPv6 is described in RFC 8200.

To enable smooth transition from IPv4 to IPv6, the features defined in this section may, in certain circumstances, also be used in connection with IPv4; see Section 2.10.20.2 (on page 559).

2.10.20.1 Addressing

IPv6 overcomes the addressing limitations of earlier versions by using 128-bit addresses instead of 32-bit addresses. The IPv6 address architecture is described in RFC 4291.

There are three kinds of IPv6 address:

Unicast
Identifies a single interface.

A unicast address can be global, link-local (designed for use on a single link), or site-local (designed for systems not connected to the Internet). Link-local and site-local addresses need not be globally unique.

Anycast
Identifies a set of interfaces such that a packet sent to the address can be delivered to any member of the set.

An anycast address is similar to a unicast address; the nodes to which an anycast address is assigned need to be explicitly configured to know that it is an anycast address.

Multicast
Identifies a set of interfaces such that a packet sent to the address should be delivered to every member of the set.

An application can send multicast datagrams by simply specifying an IPv6 multicast address in the *address* argument of *sendto*( ). To receive multicast datagrams, an application needs to join the multicast group (using *setsockopt*( ) with IPV6_JOIN_GROUP) and bind to the socket the UDP port on which datagrams are to be received. Some applications should also bind the multicast group address to the socket, to prevent other datagrams destined to that port from being delivered to the socket.

A multicast address can be global, node-local, link-local, site-local, or organization-local.

The following special IPv6 addresses are defined:

Unspecified
An address that is not assigned to any interface and is used to indicate the absence of an address.

Loopback
A unicast address that is not assigned to any interface and can be used by a node to send packets to itself.

Two sets of IPv6 addresses are defined to correspond to IPv4 addresses:

IPv4-compatible addresses
These are assigned to nodes that support IPv6 and can be used when traffic is ``tunneled'' through IPv4.

IPv4-mapped addresses
These are used to represent IPv4 addresses in IPv6 address format; see Section 2.10.20.2.

The unspecified address and the loopback address shall not be treated as IPv4-compatible addresses.

2.10.20.2 *Compatibility with IPv4*

The API provides the ability for IPv6 applications to interoperate with applications using IPv4, by using IPv4-mapped IPv6 addresses. These addresses can be generated automatically by the *getaddrinfo*( ) function when the specified host has only IPv4 addresses.

Applications can use AF_INET6 sockets to open TCP connections to IPv4 nodes, or send UDP packets to IPv4 nodes, by simply encoding the destination's IPv4 address as an IPv4-mapped IPv6 address, and passing that address, within a **sockaddr_in6** structure, in the *connect*( ), *sendto*( ), or *sendmsg*( ) function. When applications use AF_INET6 sockets to accept TCP connections from IPv4 nodes, or receive UDP packets from IPv4 nodes, the system shall return

the peer's address to the application in the *accept*(), *accept4*(), *recvfrom*(), *recvmsg*(), or *getpeername*() function using a **sockaddr_in6** structure encoded this way. If a node has an IPv4 address, then the implementation shall allow applications to communicate using that address via an AF_INET6 socket. In such a case, the address shall be represented at the API by the corresponding IPv4-mapped IPv6 address. Also, the implementation may allow an AF_INET6 socket bound to **in6addr_any** to receive inbound connections and packets destined to one of the node's IPv4 addresses.

An application can use AF_INET6 sockets to bind to a node's IPv4 address by specifying the address as an IPv4-mapped IPv6 address in a **sockaddr_in6** structure in the *bind*() function. For an AF_INET6 socket bound to a node's IPv4 address, the system shall return the address in the *getsockname*() function as an IPv4-mapped IPv6 address in a **sockaddr_in6** structure.

2.10.20.3 *Interface Identification*

Each local interface is assigned a unique positive integer as a numeric index. Indexes start at 1; zero is not used. There may be gaps so that there is no current interface for a particular positive index. Each interface also has a unique implementation-defined name.

2.10.20.4 *Options*

The following options apply at the IPPROTO_IPV6 level:

IPV6_JOIN_GROUP
: When set via *setsockopt*(), it joins the application to a multicast group on an interface (identified by its index) and addressed by a given multicast address, enabling packets sent to that address to be read via the socket. If the interface index is specified as zero, the system selects the interface (for example, by looking up the address in a routing table and using the resulting interface).

  An attempt to read this option using *getsockopt*() shall result in an [EOPNOTSUPP] error.

  The parameter type of this option is a pointer to an **ipv6_mreq** structure.

IPV6_LEAVE_GROUP
: When set via *setsockopt*(), it removes the application from the multicast group on an interface (identified by its index) and addressed by a given multicast address.

  An attempt to read this option using *getsockopt*() shall result in an [EOPNOTSUPP] error.

  The parameter type of this option is a pointer to an **ipv6_mreq** structure.

IPV6_MULTICAST_HOPS
: The value of this option is the hop limit for outgoing multicast IPv6 packets sent via the socket. Its possible values are the same as those of IPV6_UNICAST_HOPS. If the IPV6_MULTICAST_HOPS option is not set, a value of 1 is assumed. This option can be set via *setsockopt*() and read via *getsockopt*().

  The parameter type of this option is a pointer to an **int**. (Default value: 1)

IPV6_MULTICAST_IF
: The index of the interface to be used for outgoing multicast packets. It can be set via *setsockopt*() and read via *getsockopt*(). If the interface index is specified as zero, the system selects the interface (for example, by looking up the address in a routing table and using the resulting interface).

  The parameter type of this option is a pointer to an **unsigned int**. (Default value: 0)

IPV6_MULTICAST_LOOP

This option controls whether outgoing multicast packets should be delivered back to the local application when the sending interface is itself a member of the destination multicast group. If it is set to 1 they are delivered. If it is set to 0 they are not. Other values result in an [EINVAL] error. This option can be set via *setsockopt*( ) and read via *getsockopt*( ).

The parameter type of this option is a pointer to an **unsigned int** which is used as a Boolean value. (Default value: 1)

IPV6_UNICAST_HOPS

The value of this option is the hop limit for outgoing unicast IPv6 packets sent via the socket. If the option is not set, or is set to –1, the system selects a default value. Attempts to set a value less than –1 or greater than 255 shall result in an [EINVAL] error. This option can be set via *setsockopt*( ) and read via *getsockopt*( ).

The parameter type of this option is a pointer to an **int**. (Default value: Unspecified)

IPV6_V6ONLY

This socket option restricts AF_INET6 sockets to IPv6 communications only. AF_INET6 sockets may be used for both IPv4 and IPv6 communications. Some applications may want to restrict their use of an AF_INET6 socket to IPv6 communications only. For these applications, the IPv6_V6ONLY socket option is defined. When this option is turned on, the socket can be used to send and receive IPv6 packets only. This is an IPPROTO_IPV6-level option.

The parameter type of this option is a pointer to an **int** which is used as a Boolean value. (Default value: 0)

An [EOPNOTSUPP] error shall result if IPV6_JOIN_GROUP or IPV6_LEAVE_GROUP is used with *getsockopt*( ).

2.10.20.5 Headers

The symbolic constant AF_INET6 is defined in the **<sys/socket.h>** header to identify the IPv6 Internet address family. See XBD Chapter 14 (on page 221).

The **sockaddr_storage** structure defined in **<sys/socket.h>** shall be large enough to accommodate a **sockaddr_in6** structure (see the **<netinet/in.h>** header defined in XBD Chapter 14, on page 221) and shall be aligned at an appropriate boundary so that pointers to it can be cast as pointers to **sockaddr_in6** structures and used to access the fields of those structures without alignment problems. When a **sockaddr_storage** structure is cast as a **sockaddr_in6** structure, the *ss_family* field maps onto the *sin6_family* field.

The **<netinet/in.h>**, **<arpa/inet.h>**, and **<netdb.h>** headers contain other definitions used in connection with IPv6 Internet sockets; see XBD Chapter 14 (on page 221).

2.11 Data Types

2.11.1 Defined Types

All of the data types used by various functions are defined by the implementation. The following table describes some of these types. Other types referenced in the description of a function, not mentioned here, can be found in the appropriate header for that function.

Defined Type	Description
cc_t	Type used for terminal special characters.
clock_t	Integer or real-floating type used for processor times, as defined in the ISO C standard.
clockid_t	Used for clock ID type in some timer functions.
dev_t	Integer type used for device numbers.
DIR	Type representing a directory stream.
div_t	Structure type returned by the *div*( ) function.
FILE	Structure containing information about a file.
glob_t	Structure type used in pathname pattern matching.
fpos_t	Type containing all information needed to specify uniquely every position within a file.
gid_t	Integer type used for group IDs.
iconv_t	Type used for conversion descriptors.
id_t	Integer type used as a general identifier; can be used to contain at least the largest of a **pid_t**, **uid_t**, or **gid_t**.
ino_t	Unsigned integer type used for file serial numbers.
key_t	Arithmetic type used for XSI interprocess communication.
ldiv_t	Structure type returned by the *ldiv*( ) function.
mode_t	Integer type used for file attributes.
mqd_t	Used for message queue descriptors.
nfds_t	Integer type used for the number of file descriptors.
nlink_t	Integer type used for link counts.
off_t	Signed integer type used for file sizes.
pid_t	Signed integer type used for process and process group IDs.
pthread_attr_t	Used to identify a thread attribute object.
pthread_cond_t, cnd_t	Used for condition variables.
pthread_condattr_t	Used to identify a condition attribute object.
pthread_key_t, tss_t	Used for thread-specific data keys.
pthread_mutex_t, mtx_t	Used for mutexes.
pthread_mutexattr_t	Used to identify a mutex attribute object.
pthread_once_t, once_flag	Used for dynamic package initialization.
pthread_rwlock_t	Used for read-write locks.
pthread_rwlockattr_t	Used for read-write lock attributes.
pthread_t, thrd_t	Used to identify a thread.
ptrdiff_t	Signed integer type of the result of subtracting two pointers.
reclen_t	Unsigned integer type used for directory entry lengths.
regex_t	Structure type used in regular expression matching.
regmatch_t	Structure type used in regular expression matching.
rlim_t	Unsigned integer type used for limit values, to which objects of type **int** and **off_t** can be cast without loss of value.
sem_t	Type used in performing semaphore operations.
sig_atomic_t	Possibly volatile-qualified integer type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts.
sigset_t	Integer or structure type of an object used to represent sets

Defined Type	Description
	of signals.
size_t	Unsigned integer type used for size of objects.
speed_t	Type used for terminal baud rates.
ssize_t	Signed integer type used for a count of bytes or an error indication.
suseconds_t	Signed integer type used for time in microseconds.
tcflag_t	Type used for terminal modes.
time_t	Integer type used for time in seconds, as defined in the ISO C standard.
timer_t	Used for timer ID returned by the *timer_create*( ) function.
uid_t	Integer type used for user IDs.
va_list	Type used for traversing variable argument lists.
wchar_t	Integer type whose range of values can represent distinct codes for all members of the largest extended character set specified by the supported locales.
wctype_t	Scalar type which represents a character class descriptor.
wint_t	Integer type capable of storing any valid value of **wchar_t** or WEOF.
wordexp_t	Structure type used in word expansion.

2.11.2 The char Type

The type **char** is defined as a single byte; see XBD Chapter 3 (on page 31) (Byte and Character).

2.12 Status Information

Status information is data associated with a process detailing a change in the state of the process. It shall consist of:

- The state the process transitioned into (*stopped*, *continued*, or *terminated*)
- The information necessary to populate the **siginfo_t** structure provided by *waitid*( )
- If the new state is *terminated*:
  - — The low-order 8 bits of the status argument that the process passed to *_Exit*( ), *_exit*( ), or *exit*( ), or the low-order 8 bits of the value the process returned from *main*( )

    Note that these 8 bits are part of the complete value that is used to set the *si_status* member of the **siginfo_t** structure provided by *waitid*( )
  - — Whether the process terminated due to the receipt of a signal that was not caught and, if so, the number of the signal that caused the termination of the process
- If the new state is *stopped*:
  - — The number of the signal that caused the process to stop

A process might not have any status information (such as immediately after a process has started).

Status information for a process shall be generated (made available to the parent process) when the process stops, continues, or terminates except in the following case:

- If the parent process sets the action for the SIGCHLD signal to SIG_IGN, or if the parent sets the SA_NOCLDWAIT flag for the SIGCHLD signal action, process termination shall not generate new status information but shall cause any existing status information for the process to be discarded.

If new status information is generated, and the process already had status information, the existing status information shall be discarded and replaced with the new status information.

Only the process' parent process can obtain the process' status information. The parent obtains a child's status information by calling *wait*( ), *waitid*( ), or *waitpid*( ). Except when *waitid*( ) is called with the WNOWAIT flag set in the *options* argument, the status information obtained by a wait function shall be consumed (discarded) by that wait function; no two calls to *wait*( ), *waitid*( ) (without WNOWAIT), or *waitpid*( ) shall obtain the same status information.

When status information becomes available to the parent process and more than one thread in the parent process is waiting for the status information (blocked in a call to *wait*( ), *waitid*( ), or *waitpid*( ) with arguments that would match the status information):

- If none of the matching threads is in a call to *waitid*( ) with the WNOWAIT flag set in the *options* argument, the thread that obtains the status information is unspecified.
- Otherwise (at least one of the matching threads is in a call to *waitid*( ) with the WNOWAIT flag set), the matching thread or threads that obtain the status information is unspecified except that at least one of the matching threads shall obtain the status information and at most one of the matching threads that are not calling *waitid*( ) with the WNOWAIT flag set shall obtain it.

20230 *Chapter 3*

20231 # *System Interfaces*

20232 This chapter describes the functions, macros, and external variables to support applications
20233 portability at the C-language source level.

NAME

CMPLX, CMPLXF, CMPLXL — make a complex value

SYNOPSIS

```
#include <complex.h>

double complex      CMPLX(double x, double y);
float complex       CMPLXF(float x, float y);
long double complex CMPLXL(long double x, long double y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The CMPLX macros shall expand to an expression of the specified complex type, with the real part having the (converted) value of *x* and the imaginary part having the (converted) value of *y*. The resulting expression shall be suitable for use as an initializer for an object with static or thread storage duration, provided both arguments are likewise suitable.

RETURN VALUE

The CMPLX macros return the complex value $x + i\,y$ (where i is the imaginary unit).

These macros shall behave as if the implementation supported imaginary types and the definitions were:

```
#define CMPLX(x, y)  ((double complex)((double)(x) + \
                      _Imaginary_I * (double)(y)))
#define CMPLXF(x, y) ((float complex)((float)(x) + \
                      _Imaginary_I * (float)(y)))
#define CMPLXL(x, y) ((long double complex)((long double)(x) + \
                      _Imaginary_I * (long double)(y)))
```

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

20273 **NAME**

20274 FD_CLR — macros for synchronous I/O multiplexing

20275 **SYNOPSIS**

20276 `#include <sys/select.h>`

20277 `void FD_CLR(int` *`fd`*`, fd_set *`*`fdset`*`);`
20278 `int FD_ISSET(int` *`fd`*`, fd_set *`*`fdset`*`);`
20279 `void FD_SET(int` *`fd`*`, fd_set *`*`fdset`*`);`
20280 `void FD_ZERO(fd_set *`*`fdset`*`);`

20281 **DESCRIPTION**

20282 Refer to *pselect( )*.

NAME

_Exit, _exit — terminate a process

SYNOPSIS

```
#include <stdlib.h>

_Noreturn void _Exit(int status);

#include <unistd.h>

_Noreturn void _exit(int status);
```

DESCRIPTION

CX For *_Exit*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The value of *status* may be 0, EXIT_SUCCESS, EXIT_FAILURE, or any other value, though only the least significant 8 bits (that is, *status* & 0377) shall be available from *wait*( ) and *waitpid*( ); the full value shall be available from *waitid*( ) and in the **siginfo_t** passed to a signal handler for SIGCHLD.

CX The *_Exit*( ) and *_exit*( ) functions shall be functionally equivalent.

CX The *_Exit*( ) and *_exit*( ) functions shall not call functions registered with *atexit*( ) nor CX *at_quick_exit*( ), nor any registered signal handlers. Open streams shall not be flushed. Whether open streams are closed (without flushing) is implementation-defined. Finally, the calling process shall be terminated with the consequences described below.

Consequences of Process Termination

CX Process termination caused by any reason shall have the following consequences:

Note: These consequences are all extensions to the ISO C standard and are not further CX shaded. However, functionality relating to the XSI option is shaded.

- All of the file descriptors, directory streams, conversion descriptors, and message catalog descriptors open in the calling process shall be closed.
- XSI If the parent process of the calling process has set its SA_NOCLDWAIT flag or has set the action for the SIGCHLD signal to SIG_IGN:
  - — The process' status information (see Section 2.12, on page 563), if any, shall be discarded.
  - — The lifetime of the calling process shall end immediately. If SA_NOCLDWAIT is set, it is implementation-defined whether a SIGCHLD signal is sent to the parent process.
  - — If a thread in the parent process of the calling process is blocked in *wait*( ), *waitpid*( ), or *waitid*( ), and the parent process has no remaining child processes in the set of waited-for children, the *wait*( ), *waitid*( ), or *waitpid*( ) function shall fail and set *errno* to [ECHILD].

  Otherwise:

  - — Status information (see Section 2.12, on page 563) shall be generated.
  - — The calling process shall be transformed into a zombie process. Its status information shall be made available to the parent process until the process' lifetime ends.

— The process' lifetime shall end once its parent obtains the process' status information via a currently-blocked or future call to *wait*( ), *waitid*( ) (without WNOWAIT), or *waitpid*( ).

— If one or more threads in the parent process of the calling process is blocked in a call to *wait*( ), *waitid*( ), or *waitpid*( ) awaiting termination of the process, one (or, if any are calling *waitid*( ) with WNOWAIT, possibly more) of these threads shall obtain the process' status information as specified in Section 2.12 (on page 563) and become unblocked.

— A SIGCHLD shall be sent to the parent process.

- Termination of a process does not directly terminate its children. The sending of a SIGHUP signal as described below indirectly terminates children in some circumstances.
- The parent process ID of all of the existing child processes and zombie processes of the calling process shall be set to the process ID of an implementation-defined system process. That is, these processes shall be inherited by a special system process.
- XSI Each attached shared-memory segment is detached and the value of *shm_nattch* (see *shmget*( )) in the data structure associated with its shared memory ID shall be decremented by 1.
- XSI For each semaphore for which the calling process has set a *semadj* value (see *semop*( )), that value shall be added to the *semval* of the specified semaphore.
- If the process is a controlling process, the SIGHUP signal shall be sent to each process in the foreground process group of the controlling terminal belonging to the calling process.
- If the process is a controlling process, the controlling terminal associated with the session shall be disassociated from the session, allowing it to be acquired by a new controlling process.
- If the exit of the process causes a process group to become orphaned, and if any member of the newly-orphaned process group is stopped, then a SIGHUP signal followed by a SIGCONT signal shall be sent to each process in the newly-orphaned process group.
- All open named semaphores in the calling process shall be closed as if by appropriate calls to *sem_close*( ). All open unnamed semaphores in the calling process shall be destroyed as if by appropriate calls to *sem_destroy*( ).
- ML Any memory locks established by the process via calls to *mlockall*( ) or *mlock*( ) shall be removed. If locked pages in the address space of the calling process are also mapped into the address spaces of other processes and are locked by those processes, the locks established by the other processes shall be unaffected by the call by this process to *_Exit*( ) or *_exit*( ).
- Memory mappings that were created in the process shall be unmapped before the process is destroyed.
- TYM Any blocks of typed memory that were mapped in the calling process shall be unmapped, as if *munmap*( ) was implicitly called to unmap them.
- MSG All open message queue descriptors in the calling process shall be closed as if by appropriate calls to *mq_close*( ).
- Any outstanding cancelable asynchronous I/O operations may be canceled. Those asynchronous I/O operations that are not canceled shall complete as if the *_Exit*( ) or *_exit*( ) operation had not yet occurred, but any associated signal notifications shall be

suppressed. The *_Exit*( ) or *_exit*( ) operation may block awaiting such I/O completion. Whether any I/O is canceled, and which I/O may be canceled upon *_Exit*( ) or *_exit*( ), is implementation-defined.

- Threads terminated by a call to *_Exit*( ) or *_exit*( ) shall not invoke their cancellation cleanup handlers or per-thread data destructors.

RETURN VALUE

These functions do not return.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Normally applications should use *exit*( ) rather than *_Exit*( ) or *_exit*( ).

Exit statuses of 126 and greater are ambiguous in certain circumstances because they have special meanings in the shell (see Section 2.8.2 (on page 2499) and the EXIT STATUS section of *sh*).

RATIONALE

Process Termination

Early proposals drew a distinction between normal and abnormal process termination. Abnormal termination was caused only by certain signals and resulted in implementation-defined ``actions'', as discussed below. Subsequent proposals distinguished three types of termination: *normal termination* (as in the current specification), *simple abnormal termination*, and *abnormal termination with actions*. Again the distinction between the two types of abnormal termination was that they were caused by different signals and that implementation-defined actions would result in the latter case. Given that these actions were completely implementation-defined, the early proposals were only saying when the actions could occur and how their occurrence could be detected, but not what they were. This was of little or no use to conforming applications, and thus the distinction is not made in this volume of POSIX.1-2024.

The implementation-defined actions usually include, in most historical implementations, the creation of a file named **core** in the current working directory of the process. This file contains an image of the memory of the process, together with descriptive information about the process, perhaps sufficient to reconstruct the state of the process at the receipt of the signal.

There is a potential security problem in creating a **core** file if the process was set-user-ID and the current user is not the owner of the program, if the process was set-group-ID and none of the user's groups match the group of the program, or if the user does not have permission to write in the current directory. In this situation, an implementation either should not create a **core** file or should make it unreadable by the user.

Despite the silence of this volume of POSIX.1-2024 on this feature, applications are advised not to create files named **core** because of potential conflicts in many implementations. Some implementations use a name other than **core** for the file; for example, by appending the process ID to the filename.

Terminating a Process

It is important that the consequences of process termination as described occur regardless of whether the process called *_exit*( ) (perhaps indirectly through *exit*( )) or instead was terminated due to a signal or for some other reason. Note that in the specific case of *exit*( ) this means that the *status* argument to *exit*( ) is treated in the same way as the *status* argument to *_exit*( ).

A language other than C may have other termination primitives than the C-language *exit*( ) function, and programs written in such a language should use its native termination primitives, but those should have as part of their function the behavior of *_exit*( ) as described. Implementations in languages other than C are outside the scope of this version of this volume of POSIX.1-2024, however.

As required by the ISO C standard, using **return** from *main*( ) has the same behavior (other than with respect to language scope issues) as calling *exit*( ) with the returned value. Reaching the end of the *main*( ) function has the same behavior as calling *exit*(0).

A value of zero (or EXIT_SUCCESS, which is required to be zero) for the argument *status* conventionally indicates successful termination. This corresponds to the specification for *exit*( ) in the ISO C standard. The convention is followed by utilities such as *make* and various shells, which interpret a zero status from a child process as success. For this reason, applications should not call *exit*(0) or *_exit*(0) when they terminate unsuccessfully; for example, in signal-catching functions.

Historically, the implementation-defined process that inherits children whose parents have terminated without waiting on them is called *init* and has a process ID of 1.

The sending of a SIGHUP to the foreground process group when a controlling process terminates corresponds to somewhat different historical implementations. In System V, the kernel sends a SIGHUP on termination of (essentially) a controlling process. In 4.2 BSD, the kernel does not send SIGHUP in a case like this, but the termination of a controlling process is usually noticed by a system daemon, which arranges to send a SIGHUP to the foreground process group with the *vhangup*( ) function. However, in 4.2 BSD, due to the behavior of the shells that support job control, the controlling process is usually a shell with no other processes in its process group. Thus, a change to make *_exit*( ) behave this way in such systems should not cause problems with existing applications.

The termination of a process may cause a process group to become orphaned in either of two ways. The connection of a process group to its parent(s) outside of the group depends on both the parents and their children. Thus, a process group may be orphaned by the termination of the last connecting parent process outside of the group or by the termination of the last direct descendant of the parent process(es). In either case, if the termination of a process causes a process group to become orphaned, processes within the group are disconnected from their job control shell, which no longer has any information on the existence of the process group. Stopped processes within the group would languish forever. In order to avoid this problem, newly orphaned process groups that contain stopped processes are sent a SIGHUP signal and a SIGCONT signal to indicate that they have been disconnected from their session. The SIGHUP signal causes the process group members to terminate unless they are catching or ignoring SIGHUP. Under most circumstances, all of the members of the process group are stopped if any of them are stopped.

The action of sending a SIGHUP and a SIGCONT signal to members of a newly orphaned process group is similar to the action of 4.2 BSD, which sends SIGHUP and SIGCONT to each stopped child of an exiting process. If such children exit in response to the SIGHUP, any additional descendants receive similar treatment at that time. In this volume of POSIX.1-2024, the signals are sent to the entire process group at the same time. Also, in this volume of

POSIX.1-2024, but not in 4.2 BSD, stopped processes may be orphaned, but may be members of a process group that is not orphaned; therefore, the action taken at *_exit*( ) must consider processes other than child processes.

It is possible for a process group to be orphaned by a call to *setpgid*( ) or *setsid*( ), as well as by process termination. This volume of POSIX.1-2024 does not require sending SIGHUP and SIGCONT in those cases, because, unlike process termination, those cases are not caused accidentally by applications that are unaware of job control. An implementation can choose to send SIGHUP and SIGCONT in those cases as an extension; such an extension must be documented as required in **<signal.h>**.

The ISO/IEC 9899:1999 standard added the *_Exit*( ) function that results in immediate program termination without triggering signals or *atexit*( )-registered functions. In POSIX.1-2024, this is equivalent to the *_exit*( ) function.

FUTURE DIRECTIONS

None.

SEE ALSO

at_quick_exit( ), *atexit*( ), *exit*( ), *mlock*( ), *mlockall*( ), *mq_close*( ), *munmap*( ), *quick_exit*( ), *sem_close*( ), *semop*( ), *setpgid*( ), *setsid*( ), *shmget*( ), *wait*( ), *waitid*( )

XBD **<stdlib.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Interactions with the SA_NOCLDWAIT flag and SIGCHLD signal are further clarified.

The values of *status* from *exit*( ) are better described.

Issue 6

Extensions beyond the ISO C standard are marked.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for typed memory.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The *_Exit*( ) function is included.
- The DESCRIPTION is updated.

The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

References to the *wait3*( ) function are removed.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/16 is applied, correcting grammar in the DESCRIPTION.

Issue 7

Austin Group Interpretation 1003.1-2001 #031 is applied, separating these functions from the *exit*( ) function.

Austin Group Interpretation 1003.1-2001 #085 is applied, clarifying the text regarding flushing of streams and closing of temporary files.

Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, and

Semaphores options is moved to the Base.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0033 [594] and XSH/TC2-2008/0034 [594,690] are applied.

Issue 8

Austin Group Defect 368 is applied, adding a requirement for unnamed semaphores to be destroyed.

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1629 is applied, changing the APPLICATION USAGE section.

20508 **NAME**

20509 _Fork — create a new process

20510 **SYNOPSIS**

20511 ```
#include <unistd.h>
```

20512 ```
pid_t _Fork(void);
```

20513 **DESCRIPTION**

20514 Refer to *fork( )*.

NAME

a64l, l64a — convert between a 32-bit integer and a radix-64 ASCII string

SYNOPSIS

XSI
```
#include <stdlib.h>

long a64l(const char *s);
char *l64a(long value);
```

DESCRIPTION

These functions maintain numbers stored in radix-64 ASCII characters. This is a notation by which 32-bit integers can be represented by up to six characters; each character represents a digit in radix-64 notation. If the type **long** contains more than 32 bits, only the low-order 32 bits shall be used for these operations.

The characters used to represent digits are `'.'` (dot) for 0, `'/'` for 1, `'0'` through `'9'` for [2,11], `'A'` through `'Z'` for [12,37], and `'a'` through `'z'` for [38,63].

The *a64l*( ) function shall take a pointer to a radix-64 representation, in which the first digit is the least significant, and return the corresponding **long** value. If the string pointed to by *s* contains more than six characters, *a64l*( ) shall use the first six. If the first six characters of the string contain a null terminator, *a64l*( ) shall use only characters preceding the null terminator. The *a64l*( ) function shall scan the character string from left to right with the least significant digit on the left, decoding each character as a 6-bit radix-64 number. If the type **long** contains more than 32 bits, the resulting value is sign-extended. The behavior of *a64l*( ) is unspecified if *s* is a null pointer or the string pointed to by *s* was not generated by a previous call to *l64a*( ).

The *l64a*( ) function shall take a **long** argument and return a pointer to the corresponding radix-64 representation. The behavior of *l64a*( ) is unspecified if *value* is negative.

The value returned by *l64a*( ) may be a pointer into a static buffer. Subsequent calls to *l64a*( ) may overwrite the buffer.

The *l64a*( ) function need not be thread-safe.

RETURN VALUE

Upon successful completion, *a64l*( ) shall return the **long** value resulting from conversion of the input string. If a string pointed to by *s* is an empty string, *a64l*( ) shall return 0L.

The *l64a*( ) function shall return a pointer to the radix-64 representation. If *value* is 0L, *l64a*( ) shall return a pointer to an empty string.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

If the type **long** contains more than 32 bits, the result of *a64l*(*l64a*(*x*)) is *x* in the low-order 32 bits.

RATIONALE

This is not the same encoding as used by either encoding variant of the *uuencode* utility.

FUTURE DIRECTIONS

None.

SEE ALSO

strtoul( )

XBD **<stdlib.h>**

XCU *uuencode*

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

A note indicating that the *l64a*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

NAME

abort — generate an abnormal process abort

SYNOPSIS

```
#include <stdlib.h>

_Noreturn void abort(void);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *abort*( ) function shall cause abnormal process termination to occur, unless a SIGABRT signal that it generates is caught and the signal handler does not return.

CX The abnormal termination processing shall include the default actions defined for SIGABRT and may include an attempt to effect *fclose*( ) on all open streams.

CX The SIGABRT signal shall be sent to the calling thread as if by means of *raise*( ) with the CX argument SIGABRT. If this signal does not terminate the process (for example, if the signal is caught and the handler returns), *abort*( ) may change the disposition of SIGABRT to SIG_DFL and send the signal (in the same way) again. If a second signal is sent and it does not terminate the process, the behavior is unspecified, except that the *abort*( ) call shall not return.

CX The status made available to *wait*( ), *waitid*( ), or *waitpid*( ) by *abort*( ) shall be that of a process terminated by the SIGABRT signal. The *abort*( ) function shall override blocking or ignoring the SIGABRT signal.

RETURN VALUE

The *abort*( ) function shall not return.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Catching the signal is intended to provide the application developer with a portable means to abort processing, free from possible interference from any implementation-supplied functions.

RATIONALE

Historically, *abort*( ) has been implemented by calling other signal manipulation functions such as *raise*( ), *sigaction*( ), and *pthread_sigmask*( ). This means that its operation can be affected by concurrent actions in other threads. For example, if *abort*( ) attempts to terminate the process by calling *sigaction*( ) to change the disposition for SIGABRT to SIG_DFL and then calling *raise*( ), another thread could change the disposition in between those two calls, resulting in the process not being terminated. If this happens, the only requirement is that *abort*( ) does not return. An implementation could call those functions in a loop (which could in theory then execute indefinitely), or could terminate the process by calling *_exit*( ) (which would ensure termination but result in the wrong wait status). To avoid these issues, implementations are encouraged to implement *abort*( ) in a manner such that its operation cannot be affected by concurrent actions in other threads. For example, it could first halt the execution of all other threads, or it could terminate the process using a ``terminate as if by a signal'' system call instead of by raising (a second) SIGABRT.

The ISO/IEC 9899:1999 standard required (and the current standard still requires) the *abort*( ) function to be async-signal-safe. Since POSIX.1-2024 defers to the ISO C standard, this required a

change to the DESCRIPTION from ``shall include the effect of *fclose*( )'' to ``may include an attempt to effect *fclose*( ).''

The revised wording permits some backwards-compatibility and avoids a potential deadlock situation.

The Open Group Base Resolution bwg2002-003 is applied, removing the following XSI shaded paragraph from the DESCRIPTION:

``On XSI-conformant systems, in addition the abnormal termination processing shall include the effect of *fclose*( ) on message catalog descriptors.''

There were several reasons to remove this paragraph:

- No special processing of open message catalogs needs to be performed prior to abnormal process termination.
- The main reason to specifically mention that *abort*( ) includes the effect of *fclose*( ) on open streams is to flush output queued on the stream. Message catalogs in this context are read-only and, therefore, do not need to be flushed.
- The effect of *fclose*( ) on a message catalog descriptor is unspecified. Message catalog descriptors are allowed, but not required to be implemented using a file descriptor, but there is no mention in POSIX.1-2024 of a message catalog descriptor using a standard I/O stream FILE object as would be expected by *fclose*( ).

FUTURE DIRECTIONS

A future version of this standard may require *abort*( ) to be implemented in a manner such that its operation cannot be affected by concurrent actions in other threads.

SEE ALSO

exit( ), *kill*( ), *raise*( ), *signal*( ), *wait*( ), *waitid*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

Changes are made to the DESCRIPTION for alignment with the ISO/IEC 9899: 1999 standard.

The Open Group Base Resolution bwg2002-003 is applied.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/10 is applied, changing the DESCRIPTION of abnormal termination processing and adding to the RATIONALE section.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/9 is applied, changing ``implementation-defined functions'' to ``implementation-supplied functions'' in the APPLICATION USAGE section.

Issue 8

Austin Group Defect 906 is applied, clarifying how the behavior of *abort*( ) may be affected by concurrent actions in other threads.

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899: 2018 standard.

NAME

abs — return an integer absolute value

SYNOPSIS

```
#include <stdlib.h>

int abs(int i);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *abs*( ) function shall compute the absolute value of its integer operand, *i*. If the result cannot be represented, the behavior is undefined.

RETURN VALUE

The *abs*( ) function shall return the absolute value of its integer operand.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Since POSIX.1 requires a two's complement representation of **int**, the absolute value of the negative integer with the largest magnitude {INT_MIN} is not representable, thus *abs*(INT_MIN) is undefined.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fabs( ), *labs*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

Issue 8

Austin Group Defect 1108 is applied, changing the APPLICATION USAGE section.

20689 **NAME**

20690 accept, accept4 — accept a new connection on a socket

20691 **SYNOPSIS**

20692
```
#include <sys/socket.h>
```

20693
```
int accept(int socket, struct sockaddr *restrict address,
```
20694
```
    socklen_t *restrict address_len);
```
20695
```
int accept4(int socket, struct sockaddr *restrict address,
```
20696
```
    socklen_t *restrict address_len, int flag);
```

20697 **DESCRIPTION**

20698 The *accept*( ) function shall extract the first connection on the queue of pending connections,
20699 create a new socket with the same socket type protocol and address family as the specified
20700 socket, and allocate a new file descriptor for that socket. The file descriptor shall be allocated as
20701 described in Section 2.6 (on page 525).

20702 The *accept*( ) function takes the following arguments:

20703 *socket* Specifies a socket that was created with *socket*( ), has been bound to an address
20704 with *bind*( ), and has issued a successful call to *listen*( ).

20705 *address* Either a null pointer, or a pointer to a **sockaddr** structure where the address of
20706 the connecting socket shall be returned.

20707 *address_len* Either a null pointer, if *address* is a null pointer, or a pointer to a **socklen_t**
20708 object which on input specifies the length of the supplied **sockaddr** structure,
20709 and on output specifies the length of the address of the connecting socket.

20710 If *address* is not a null pointer, the address of the peer for the accepted connection shall be stored
20711 in the **sockaddr** structure pointed to by *address*, and the length of this address shall be stored in
20712 the object pointed to by *address_len*.

20713 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,
20714 the stored address shall be truncated.

20715 If the protocol permits connections by unbound clients, and the peer is not bound, then the
20716 value stored in the object pointed to by *address* is unspecified.

20717 If the listen queue is empty of connection requests and O_NONBLOCK is not set on the file
20718 descriptor for the socket, *accept*( ) shall block until a connection is present. If the *listen*( ) queue is
20719 empty of connection requests and O_NONBLOCK is set on the file descriptor for the socket,
20720 *accept*( ) shall fail and set *errno* to [EAGAIN] or [EWOULDBLOCK].

20721 The accepted socket cannot itself accept more connections. The original socket remains open and
20722 can accept more connections.

20723 If O_NONBLOCK is set on the file description for *socket*, it is implementation-defined whether
20724 O_NONBLOCK will be set on the file description created by *accept*( ). FD_CLOEXEC and
20725 FD_CLOFORK for the new file descriptor shall be clear, regardless of how they are currently set
20726 for *socket*.

20727 It is implementation-defined which socket options, if any, on the accepted socket will have a
20728 default value determined by a value previously customized by *setsockopt*( ) on *socket*, rather than
20729 the default value used for other new sockets.

20730 The *accept4*( ) function shall be equivalent to the *accept*( ) function, except that the state of
20731 O_NONBLOCK on the new file description, and FD_CLOEXEC and FD_CLOFORK on the
20732 returned file descriptor shall be determined solely by the *flag* argument, which can be
20733 constructed from a bitwise-inclusive OR of flags from the following list:

SOCK_CLOEXEC Atomically set the FD_CLOEXEC flag on the new file descriptor.

SOCK_CLOFORK Atomically set the FD_CLOFORK flag on the new file descriptor.

SOCK_NONBLOCK Set the O_NONBLOCK file status flag on the new file description.

Implementations may define additional flags.

RETURN VALUE

Upon successful completion, *accept*( ) and *accept4*( ) shall return the non-negative file descriptor of the accepted socket. Otherwise, –1 shall be returned, *errno* shall be set to indicate the error, and any object pointed to by *address_len* shall remain unchanged.

ERRORS

The *accept*( ) and *accept4*( ) functions shall fail if:

[EAGAIN] or [EWOULDBLOCK]
O_NONBLOCK is set for the socket file descriptor and no connections are present to be accepted.

[EBADF] The *socket* argument is not a valid file descriptor.

[ECONNABORTED]
A connection has been aborted.

[EINTR] The *accept*( ) function was interrupted by a signal that was caught before a valid connection arrived.

[EINVAL] The *socket* is not accepting connections.

[EMFILE] All file descriptors available to the process are currently open.

[ENFILE] The maximum number of file descriptors in the system are already open.

[ENOBUFS] No buffer space is available.

[ENOMEM] There was insufficient memory available to complete the operation.

[ENOTSOCK] The *socket* argument does not refer to a socket.

[EOPNOTSUPP] The socket type of the specified socket does not support accepting connections.

The *accept*( ) and *accept4*( ) functions may fail if:

[EPROTO] A protocol error has occurred.

The *accept4*( ) function may fail if:

[EINVAL] The value of the *flag* argument is invalid.

EXAMPLES

None.

APPLICATION USAGE

When a connection is available, *select*( ) indicates that the file descriptor for the socket is ready for reading.

Many socket options are described as having implementation-defined default values, which may differ according to the protocol in use by the socket. Existing practice differs on whether socket options such as SO_SNDBUF that were customized on the original listening socket will impact the corresponding option on the newly returned socket. Implementations are permitted to allow inheritance of customized settings where it makes sense, although the most portable

approach for applications is to limit *setsockopt*( ) customizations to only the accepted socket.

For AF_UNIX sockets, it is recommended that *address* points to a buffer of length greater than `sizeof(struct sockaddr_un)` which has been initialized with null bytes. That way, even if the implementation supports the use of all bytes of *sun_path* without a terminating null byte, the larger buffer guarantees that the *sun_path* member can then be passed to other interfaces that expect a null-terminated string. If no truncation occurred based on the input value of *address_len*, it is unspecified whether the returned *address_len* will be `sizeof(struct sockaddr_un)`, or merely a value at least as large as `offsetof(struct sockaddr_un, sun_path)` plus the number of non-null bytes stored in *sun_path*.

RATIONALE

The SOCK_CLOEXEC and SOCK_CLOFORK flags of *accept4*( ) are necessary to avoid a data race in multi-threaded applications. Without SOCK_CLOFORK, a file descriptor is leaked into a child process created by one thread in the window between another thread creating a file descriptor with *accept*( ) and then using *fcntl*( ) to set the FD_CLOFORK flag. Without SOCK_CLOEXEC, a file descriptor intentionally inherited by child processes is similarly leaked into an executed program if FD_CLOEXEC is not set atomically.

Two designs often used for network servers are multi-threaded servers with a pre-created pool of worker threads, where the thread that accepts the connection request hands over the new file descriptor to a worker thread for servicing, and pre-fork servers with a pre-created pool of worker processes, where the process that accepts the connection request passes the new file descriptor (for example via *sendmsg*( )) to a worker process. In both of these designs, *accept4*( ) should be used with the SOCK_CLOFORK flag set. Simpler designs are also sometimes used that do not pre-create a pool. For a multi-threaded server that creates a thread to handle each request, SOCK_CLOFORK should still be used. For a forking server that creates a child to service each request, clearly SOCK_CLOFORK cannot be used if the child is to inherit the file descriptor to be serviced, and therefore this type of server needs to use an alternative method of indicating the end of communications, for example using *shutdown*( ), to ensure the client sees end-of-file, rather than just closing the socket. Such child processes should set FD_CLOFORK on the inherited file descriptor before they attempt to start any additional child processes to avoid leakage into those children.

The SOCK_NONBLOCK flag is for convenience in avoiding additional *fcntl*( ) calls, as well as providing specific control over the O_NONBLOCK flag, since traditional implementations of *accept*( ) differ on whether O_NONBLOCK is inherited from the *socket* argument.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.6 (on page 525), *bind*( ), *connect*( ), *listen*( ), *socket*( )

XBD **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The **restrict** keyword is added to the *accept*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

Austin Group Interpretation 1003.1-2001 #044 is applied, changing the ``may fail'' [ENOBUFS] and [ENOMEM] errors to become ``shall fail'' errors.

Functionality relating to XSI STREAMS is marked obsolescent.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0018 [464] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0035 [835] and XSH/TC2-2008/0036 [836] are applied.

Issue 8

Austin Group Defects 411 and 1318 are applied, adding *accept4*( ), requiring FD_CLOEXEC and FD_CLOFORK to be clear for the file descriptor returned by *accept*( ), and clarifying the requirements for O_NONBLOCK on the file description created by *accept*( ).

Austin Group Defect 561 is applied, adding a paragraph about *sun_path* to APPLICATION USAGE.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1337 is applied, clarifying socket option default values.

Austin Group Defect 1565 is applied, changing the description of *address_len*.

20833 **NAME**

20834 access, faccessat — determine accessibility of a file descriptor

20835 **SYNOPSIS**

20836 ```
#include <unistd.h>
```

20837 ```
int access(const char *path, int amode);
```

20838 OH ```
#include <fcntl.h>
```

20839 ```
int faccessat(int fd, const char *path, int amode, int flag);
```

20840 **DESCRIPTION**

20841 The *access*( ) function shall check the file named by the pathname pointed to by the *path*
20842 argument for accessibility according to the bit pattern contained in *amode*. The checks for
20843 accessibility (including directory permissions checked during pathname resolution) shall be
20844 performed using the real user ID in place of the effective user ID and the real group ID in place
20845 of the effective group ID.

20846 The value of *amode* is either the bitwise-inclusive OR of the access permissions to be checked
20847 (R_OK, W_OK, X_OK) or the existence test (F_OK).

20848 If any access permissions are checked, each shall be checked individually, as described in XBD
20849 Section 4.7 (on page 97), except that where that description refers to execute permission for a
20850 process with appropriate privileges, an implementation may indicate success for X_OK even if
20851 execute permission is not granted to any user.

20852 The *faccessat*( ) function, when called with a *flag* value of zero, shall be equivalent to the *access*( )
20853 function, except in the case where *path* specifies a relative path. In this case the file whose
20854 accessibility is to be determined shall be located relative to the directory associated with the file
20855 descriptor *fd* instead of the current working directory. If the access mode of the open file
20856 description associated with the file descriptor is not O_SEARCH, the function shall check
20857 whether directory searches are permitted using the current permissions of the directory
20858 underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform
20859 the check.

20860 If *faccessat*( ) is passed the special value AT_FDCWD in the *fd* parameter, the current working
20861 directory shall be used and, if flag is zero, the behavior shall be identical to a call to *access*( ).

20862 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined
20863 in **<fcntl.h>**:

20864 AT_EACCESS The checks for accessibility (including directory permissions checked during
20865 pathname resolution) shall be performed using the effective user ID and
20866 group ID instead of the real user ID and group ID as required in a call to
20867 *access*( ).

20868 **RETURN VALUE**

20869 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
20870 return –1 and set *errno* to indicate the error.

20871 **ERRORS**

20872 These functions shall fail if:

20873 [EACCES] Permission bits of the file mode do not permit the requested access, or search
20874 permission is denied on a component of the path prefix.

20875 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
20876 argument.

20877 [ENAMETOOLONG]
20878 The length of a component of a pathname is longer than {NAME_MAX}.

20879 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

20880 [ENOTDIR] A component of the path prefix names an existing file that is neither a
20881 directory nor a symbolic link to a directory, or the *path* argument contains at
20882 least one non-<slash> character and ends with one or more trailing <slash>
20883 characters and the last pathname component names an existing file that is
20884 neither a directory nor a symbolic link to a directory.

20885 [EROFS] Write access is requested for a file on a read-only file system.

20886 The *faccessat*( ) function shall fail if:

20887 [EACCES] The access mode of the open file description associated with *fd* is not
20888 O_SEARCH and the permissions of the directory underlying *fd* do not permit
20889 directory searches.

20890 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
20891 neither AT_FDCWD nor a valid file descriptor open for reading or searching.

20892 [ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated
20893 with a non-directory file.

20894 These functions may fail if:

20895 [EINVAL] The value of the *amode* argument is invalid.

20896 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
20897 resolution of the *path* argument.

20898 [ENAMETOOLONG]
20899 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
20900 symbolic link produced an intermediate result with a length that exceeds
20901 {PATH_MAX}.

20902 [ETXTBSY] Write access is requested for a pure procedure (shared text) file that is being
20903 executed.

20904 The *faccessat*( ) function may fail if:

20905 [EINVAL] The value of the *flag* argument is not valid.

20906 **EXAMPLES**

20907 **Testing for the Existence of a File**

20908 The following example tests whether a file named **myfile** exists in the **/tmp** directory.

```
20909 #include <unistd.h>
20910 ...
20911 int result;
20912 const char *pathname = "/tmp/myfile";
20913 result = access (pathname, F_OK);
```


APPLICATION USAGE

Use of these functions is discouraged since by the time the returned information is acted upon, it is out-of-date. (That is, acting upon the information always leads to a time-of-check-to-time-of-use race condition.) An application should instead attempt the action itself and handle the [EACCES] error that occurs if the file is not accessible (with a change of effective user and group IDs beforehand, and perhaps a change back afterwards, in the case where *access*( ) or *faccessat*( ) without AT_EACCES would have been used.)

Historically, one of the uses of *access*( ) was in set-user-ID root programs to check whether the user running the program had access to a file. This relied on ``super-user'' privileges which were granted based on the effective user ID being zero, so that when *access*( ) used the real user ID to check accessibility those privileges were not taken into account. On newer systems where privileges can be assigned which have no association with user or group IDs, if a program with such privileges calls *access*( ), the change of IDs has no effect on the privileges and therefore they are taken into account in the accessibility checks. Thus, *access*( ) (and *faccessat*( ) with flag zero) cannot be used for this historical purpose in such programs. Likewise, if a system provides any additional or alternate file access control mechanisms that are not user ID-based, they will still be taken into account.

If a relative pathname is used, no account is taken of whether the current directory (or the directory associated with the file descriptor *fd*) is accessible via any absolute pathname. Applications using *access*( ), or *faccessat*( ) without AT_EACCES, may consequently act as if the file would be accessible to a user with the real user ID and group ID of the process when such a user would not in practice be able to access the file because access would be denied at some point above the current directory (or the directory associated with the file descriptor *fd*) in the file hierarchy.

If *access*( ) or *faccessat*( ) is used with W_OK to check for write access to a directory which has the S_ISVTX bit set, a return value indicating the directory is writable can be misleading since some operations on files in the directory would not be permitted based on the ownership of those files (see XBD Section 4.5, on page 96).

Additional values of *amode* other than the set defined in the description may be valid; for example, if a system has extended access controls.

The use of the AT_EACCESS value for *flag* enables functionality not available in *access*( ).

RATIONALE

In early proposals, some inadequacies in the *access*( ) function led to the creation of an *eaccess*( ) function because:

1. Historical implementations of *access*( ) do not test file access correctly when the process' real user ID is superuser. In particular, they always return zero when testing execute permissions without regard to whether the file is executable.
2. The superuser has complete access to all files on a system. As a consequence, programs started by the superuser and switched to the effective user ID with lesser privileges cannot use *access*( ) to test their file access permissions.

However, the historical model of *eaccess*( ) does not resolve problem (1), so this volume of POSIX.1-2024 now allows *access*( ) to behave in the desired way because several implementations have corrected the problem. It was also argued that problem (2) is more easily solved by using *open*( ), *chdir*( ), or one of the *exec* functions as appropriate and responding to the error, rather than creating a new function that would not be as reliable. Therefore, *eaccess*( ) is not included in this volume of POSIX.1-2024.

The sentence concerning appropriate privileges and execute permission bits reflects the two

possibilities implemented by historical implementations when checking superuser access for X_OK.

New implementations are discouraged from returning X_OK unless at least one execution permission bit is set.

The purpose of the *faccessat*( ) function is to enable the checking of the accessibility of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *access*( ), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *faccessat*( ) function it can be guaranteed that the file tested for accessibility is located relative to the desired directory.

FUTURE DIRECTIONS

These functions may be formally deprecated (for example, by shading them OB) in a future version of this standard.

SEE ALSO

chmod( ), *fstatat*( )

XBD Section 4.7 (on page 97), **<fcntl.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [ETXTBSY] optional error condition is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

Issue 7

Austin Group Interpretations 1003.1-2001 #046 and #143 are applied.

The *faccessat*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made to allow a directory to be opened for searching.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0019 [461], XSH/TC1-2008/0020 [324], XSH/TC1-2008/0021 [278], XSH/TC1-2008/0022 [278], and XSH/TC1-2008/0023 [291] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0037 [873], XSH/TC2-2008/0038 [591], XSH/TC2-2008/0039 [838], XSH/TC2-2008/0040 [817], XSH/TC2-2008/0041 [487], XSH/TC2-2008/0042 [838], XSH/TC2-2008/0043 [817], and XSH/TC2-2008/0044 [838] are applied.

21000 **NAME**

21001 acos, acosf, acosl — arc cosine functions

21002 **SYNOPSIS**

21003
```
#include <math.h>
```

21004
```
double acos(double x);
```
21005
```
float acosf(float x);
```
21006
```
long double acosl(long double x);
```

21007 **DESCRIPTION**

21008 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21009 conflict between the requirements described here and the ISO C standard is unintentional. This
21010 volume of POSIX.1-2024 defers to the ISO C standard.

21011 These functions shall compute the principal value of the arc cosine of their argument *x*. The
21012 value of *x* should be in the range [–1,1].

21013 An application wishing to check for error situations should set *errno* to zero and call
21014 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
21015 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
21016 zero, an error has occurred.

21017 **RETURN VALUE**

21018 Upon successful completion, these functions shall return the arc cosine of *x*, in the range [0,π]
21019 radians.

21020 MX For finite values of *x* not in the range [–1,1], a domain error shall occur, and either a NaN (if
21021 supported), or an implementation-defined value shall be returned.

21022 MX If *x* is NaN, a NaN shall be returned.

21023 If *x* is +1, +0 shall be returned.

21024 If *x* is ±Inf, a domain error shall occur, and a NaN shall be returned.

21025 **ERRORS**

21026 These functions shall fail if:

21027 MX Domain Error The *x* argument is finite and is not in the range [–1,1], or is ±Inf.

21028 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
21029 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
21030 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
21031 shall be raised.

21032 **EXAMPLES**

21033 None.

21034 **APPLICATION USAGE**

21035 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
21036 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

21037 **RATIONALE**

21038 None.

21039 **FUTURE DIRECTIONS**

21040 None.

21041 **SEE ALSO**

21042 *cos*( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( )

21043 XBD Section 4.23 (on page 109), **<math.h>**

21044 **CHANGE HISTORY**

21045 First released in Issue 1. Derived from Issue 1 of the SVID.

21046 **Issue 5**

21047 The DESCRIPTION is updated to indicate how an application should check for an error. This
21048 text was previously published in the APPLICATION USAGE section.

21049 **Issue 6**

21050 The *acosf*( ) and *acosl*( ) functions are added for alignment with the ISO/IEC 9899: 1999 standard.

21051 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
21052 revised to align with the ISO/IEC 9899: 1999 standard.

21053 IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are
21054 marked.

21055 **Issue 7**

21056 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0024 [320] is applied.

21057 **NAME**

21058 acosh, acoshf, acoshl — inverse hyperbolic cosine functions

21059 **SYNOPSIS**

21060 ```
#include <math.h>
```

21061 ```
double acosh(double x);
```
21062 ```
float acoshf(float x);
```
21063 ```
long double acoshl(long double x);
```

21064 **DESCRIPTION**

21065 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21066 conflict between the requirements described here and the ISO C standard is unintentional. This
21067 volume of POSIX.1-2024 defers to the ISO C standard.

21068 These functions shall compute the inverse hyperbolic cosine of their argument *x*.

21069 An application wishing to check for error situations should set *errno* to zero and call
21070 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
21071 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
21072 zero, an error has occurred.

21073 **RETURN VALUE**

21074 Upon successful completion, these functions shall return the inverse hyperbolic cosine of their
21075 argument.

21076 MX For finite values of $x < 1$, a domain error shall occur, and either a NaN (if supported), or an
21077 implementation-defined value shall be returned.

21078 MX If *x* is NaN, a NaN shall be returned.

21079 If *x* is +1, +0 shall be returned.

21080 If *x* is +Inf, +Inf shall be returned.

21081 If *x* is –Inf, a domain error shall occur, and a NaN shall be returned.

21082 **ERRORS**

21083 These functions shall fail if:

21084 MX Domain Error The *x* argument is finite and less than +1.0, or is –Inf.

21085 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
21086 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
21087 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
21088 shall be raised.

21089 **EXAMPLES**

21090 None.

21091 **APPLICATION USAGE**

21092 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
21093 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

21094 **RATIONALE**

21095 None.

21096 **FUTURE DIRECTIONS**

21097 None.

21098 **SEE ALSO**

21099 *cosh*( ), *feclearexcept*( ), *fetestexcept*( )

21100 XBD Section 4.23 (on page 109), **<math.h>**

21101 **CHANGE HISTORY**

21102 First released in Issue 4, Version 2.

21103 **Issue 5**

21104 Moved from X/OPEN UNIX extension to BASE.

21105 **Issue 6**

21106 The *acosh*( ) function is no longer marked as an extension.

21107 The *acoshf*( ) and *acoshl*( ) functions are added for alignment with the ISO/IEC 9899:1999
21108 standard.

21109 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
21110 revised to align with the ISO/IEC 9899:1999 standard.

21111 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
21112 marked.

21113 **Issue 7**

21114 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0025 [320] is applied.

21115 **NAME**

21116 acosl — arc cosine functions

21117 **SYNOPSIS**

21118 `#include <math.h>`

21119 `long double acosl(long double x);`

21120 **DESCRIPTION**

21121 Refer to *acos*( ).

21122 **NAME**

21123 aio_cancel — cancel an asynchronous I/O request

21124 **SYNOPSIS**

21125 `#include <aio.h>`

21126 `int aio_cancel(int fildes, struct aiocb *aiocbp);`

21127 **DESCRIPTION**

21128 The *aio_cancel*( ) function shall attempt to cancel one or more asynchronous I/O requests
21129 currently outstanding against file descriptor *fildes*. The *aiocbp* argument points to the
21130 asynchronous I/O control block for a particular request to be canceled. If *aiocbp* is NULL, then
21131 all outstanding cancelable asynchronous I/O requests against *fildes* shall be canceled.

21132 Normal asynchronous notification shall occur for asynchronous I/O operations that are
21133 successfully canceled. If there are requests that cannot be canceled, then the normal
21134 asynchronous completion process shall take place for those requests when they are completed.

21135 For requested operations that are successfully canceled, the associated error status shall be set to
21136 [ECANCELED] and the return status shall be –1. For requested operations that are not
21137 successfully canceled, the *aiocbp* shall not be modified by *aio_cancel*( ).

21138 If *aiocbp* is not NULL, then if *fildes* does not have the same value as the file descriptor with which
21139 the asynchronous operation was initiated, unspecified results occur.

21140 Which operations are cancelable is implementation-defined.

21141 **RETURN VALUE**

21142 The *aio_cancel*( ) function shall return the value AIO_CANCELED if the requested operation(s)
21143 were canceled. The value AIO_NOTCANCELED shall be returned if at least one of the
21144 requested operation(s) cannot be canceled because it is in progress. In this case, the state of the
21145 other operations, if any, referenced in the call to *aio_cancel*( ) is not indicated by the return value
21146 of *aio_cancel*( ). The application may determine the state of affairs for these operations by using
21147 *aio_error*( ). The value AIO_ALLDONE is returned if all of the operations have already
21148 completed. Otherwise, the function shall return –1 and set *errno* to indicate the error.

21149 **ERRORS**

21150 The *aio_cancel*( ) function shall fail if:

21151 [EBADF] The *fildes* argument is not a valid file descriptor.

21152 **EXAMPLES**

21153 None.

21154 **APPLICATION USAGE**

21155 None.

21156 **RATIONALE**

21157 None.

21158 **FUTURE DIRECTIONS**

21159 None.

21160 **SEE ALSO**

21161 *aio_read*( ), *aio_write*( )

21162 XBD **<aio.h>**

21163 **CHANGE HISTORY**

21164 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

21165 **Issue 6**

21166 The [ENOSYS] error condition has been removed as stubs need not be provided if an
21167 implementation does not support the Asynchronous Input and Output option.

21168 The APPLICATION USAGE section is added.

21169 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/10 is applied, removing the words ``to the
21170 calling process'' in the RETURN VALUE section. The term was unnecessary and precluded
21171 threads.

21172 **Issue 7**

21173 The *aio_cancel*( ) function is moved from the Asynchronous Input and Output option to the Base.

21174 **NAME**

21175 aio_error — retrieve errors status for an asynchronous I/O operation

21176 **SYNOPSIS**

21177
```
#include <aio.h>
```

21178
```
int aio_error(const struct aiocb *aiocbp);
```

21179 **DESCRIPTION**

21180 The *aio_error*( ) function shall return the error status associated with the **aiocb** structure
21181 referenced by the *aiocbp* argument. The error status for an asynchronous I/O operation is the
21182 SIO *errno* value that would be set by the corresponding *read*( ), *write*( ), *fdatasync*( ), or *fsync*( )
21183 operation. If the operation has not yet completed, then the error status shall be equal to
21184 [EINPROGRESS].

21185 If the **aiocb** structure pointed to by *aiocbp* is not associated with an operation that has been
21186 scheduled, the results are undefined.

21187 **RETURN VALUE**

21188 If the asynchronous I/O operation has completed successfully, then 0 shall be returned. If the
21189 asynchronous operation has completed unsuccessfully, then the error status, as described for
21190 SIO *read*( ), *write*( ), *fdatasync*( ), and *fsync*( ), shall be returned. If the asynchronous I/O operation has
21191 not yet completed, then [EINPROGRESS] shall be returned.

21192 If the *aio_error*( ) function fails, it shall return –1 and set *errno* to indicate the error.

21193 **ERRORS**

21194 The *aio_error*( ) function may fail if:

21195 [EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose
21196 return status has not yet been retrieved.

21197 **EXAMPLES**

21198 None.

21199 **APPLICATION USAGE**

21200 None.

21201 **RATIONALE**

21202 None.

21203 **FUTURE DIRECTIONS**

21204 None.

21205 **SEE ALSO**

21206 *aio_cancel*( ), *aio_fsync*( ), *aio_read*( ), *aio_return*( ), *aio_write*( ), *close*( ), *exec*, *exit*( ), *fork*( ), *lio_listio*( ),
21207 *lseek*( ), *read*( )

21208 XBD **<aio.h>**

21209 **CHANGE HISTORY**

21210 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

21211 **Issue 6**

21212 The [ENOSYS] error condition has been removed as stubs need not be provided if an
21213 implementation does not support the Asynchronous Input and Output option.

21214 The APPLICATION USAGE section is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #045 is applied.

SD5-XSH-ERN-148 is applied.

The *aio_error*( ) function is moved from the Asynchronous Input and Output option to the Base.

NAME

aio_fsync — asynchronous file synchronization

SYNOPSIS

FSC|SIO

```
#include <aio.h>

int aio_fsync(int op, struct aiocb *aiocbp);
```

DESCRIPTION

The *aio_fsync*( ) function shall asynchronously perform a file synchronization operation, as specified by the *op* argument, for I/O operations associated with the file indicated by the file descriptor *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp* argument and queued at the time of the call to *aio_fsync*( ). The function call shall return when the synchronization request has been initiated or queued to the file or device (even when the data cannot be synchronized immediately).

SIO If *op* is O_DSYNC, all currently queued I/O operations shall be completed as if by a call to *fdatasync*( ); that is, as defined for synchronized I/O data integrity completion.

FSC If *op* is O_SYNC, all currently queued I/O operations shall be completed as if by a call to *fsync*( ); FSC SIO that is, as defined for synchronized I/O file integrity completion. If the *aio_fsync*( ) function fails, or if the operation queued by *aio_fsync*( ) fails, then outstanding I/O operations are not guaranteed to have been completed.

If *aio_fsync*( ) succeeds, then it is only the I/O that was queued at the time of the call to *aio_fsync*( ) that is guaranteed to be forced to the relevant completion state. The completion of subsequent I/O on the file descriptor is not guaranteed to be completed in a synchronized fashion.

The *aiocbp* argument refers to an asynchronous I/O control block. The *aiocbp* value may be used as an argument to *aio_error*( ) and *aio_return*( ) in order to determine the error status and return status, respectively, of the asynchronous operation while it is proceeding. When the request is queued, the error status for the operation is [EINPROGRESS]. When all data has been successfully transferred, the error status shall be reset to reflect the success or failure of the operation. If the operation does not complete successfully, the error status for the operation shall be set to indicate the error. The *aio_sigevent* member determines the asynchronous notification to occur as specified in Section 2.4.1 (on page 513) when all operations have achieved synchronized I/O completion. All other members of the structure referenced by *aiocbp* are ignored. If the control block referenced by *aiocbp* becomes an illegal address prior to asynchronous I/O completion, then the behavior is undefined.

If the *aio_fsync*( ) function fails or *aiocbp* indicates an error condition, data is not guaranteed to have been successfully transferred.

RETURN VALUE

The *aio_fsync*( ) function shall return the value 0 if the I/O operation is successfully queued; otherwise, the function shall return the value –1 and set *errno* to indicate the error.

ERRORS

The *aio_fsync*( ) function shall fail if:

[EAGAIN] The requested asynchronous operation was not queued due to temporary resource limitations.

[EBADF] The *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp* argument is not a valid file descriptor.

SIO [EINVAL] This implementation does not support synchronized I/O for this file.

FSC [EINVAL] The *aio_fildes* member of the **aiocb** structure refers to a file on which an *fsync*( ) operation is not possible.

[EINVAL] A value of *op* other than O_DSYNC or O_SYNC was specified, or O_DSYNC was specified and the implementation does not provide runtime support for the Synchronized Input and Output option, or O_SYNC was specified and the implementation does not provide runtime support for the File Synchronization option.

In the event that any of the queued I/O operations fail, *aio_fsync*( ) shall return the error condition defined for *read*( ) and *write*( ). The error is returned in the error status for the asynchronous operation, which can be retrieved using *aio_error*( ).

EXAMPLES

None.

APPLICATION USAGE

Note that even if the file descriptor is not open for writing, if there are any pending write requests on the underlying file, then that I/O will be completed prior to the return of a call to *aio_error*( ) or *aio_return*( ) indicating that the operation has completed.

See also the APPLICATION USAGE for *fdatasync*( ) and *fsync*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_error( ), *aio_return*( ), *fcntl*( ), *fdatasync*( ), *fsync*( ), *open*( ), *read*( ), *write*( )

XBD **<aio.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The APPLICATION USAGE section is added.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/11 is applied, removing the words ``to the calling process'' in the RETURN VALUE section. The term was unnecessary and precluded threads.

Issue 7

The *aio_fsync*( ) function is moved from the Asynchronous Input and Output option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0026 [98] and XSH/TC1-2008/0027 [98] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0045 [671] is applied.

21302 **Issue 8**

21303 Austin Group Defect 672 is applied, changing the APPLICATION USAGE section.

21304 **NAME**

21305 aio_read — asynchronous read from a file

21306 **SYNOPSIS**

21307 `#include <aio.h>`

21308 `int aio_read(struct aiocb *aiocbp);`

21309 **DESCRIPTION**

21310 The *aio_read*( ) function shall read *aiocbp*−>*aio_nbytes* from the file associated with
21311 *aiocbp*−>*aio_fildes* into the buffer pointed to by *aiocbp*−>*aio_buf*. The function call shall return
21312 when the read request has been initiated or queued to the file or device (even when the data
21313 cannot be delivered immediately).

21314 PIO If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted
21315 at a priority equal to a base scheduling priority minus *aiocbp*−>*aio_reqprio*. If Thread Execution
21316 Scheduling is not supported, then the base scheduling priority is that of the calling process;
21317 PIO TPS otherwise, the base scheduling priority is that of the calling thread.

21318 The *aiocbp* value may be used as an argument to *aio_error*( ) and *aio_return*( ) in order to
21319 determine the error status and return status, respectively, of the asynchronous operation while it
21320 is proceeding. If an error condition is encountered during queuing, the function call shall return
21321 without having initiated or queued the request. The requested operation takes place at the
21322 absolute position in the file as given by *aio_offset*, as if *lseek*( ) were called immediately prior to
21323 the operation with an *offset* equal to *aio_offset* and a *whence* equal to SEEK_SET. After a
21324 successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file
21325 is unspecified.

21326 The *aio_sigevent* member specifies the notification which occurs when the request is completed.

21327 The *aiocbp*−>*aio_lio_opcode* field shall be ignored by *aio_read*( ).

21328 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp*−>*aio_buf* or
21329 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O
21330 completion, then the behavior is undefined.

21331 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

21332 SIO If synchronized I/O is enabled on the file associated with *aiocbp*−>*aio_fildes*, the behavior of this
21333 function shall be according to the definitions of synchronized I/O data integrity completion and
21334 synchronized I/O file integrity completion.

21335 For any system action that changes the process memory space while an asynchronous I/O is
21336 outstanding to the address range being changed, the result of that action is undefined.

21337 For regular files, no data transfer shall occur past the offset maximum established in the open
21338 file description associated with *aiocbp*−>*aio_fildes*.

21339 **RETURN VALUE**

21340 The *aio_read*( ) function shall return the value zero if the I/O operation is successfully queued;
21341 otherwise, the function shall return the value −1 and set *errno* to indicate the error.

21342 **ERRORS**

21343 The *aio_read*( ) function shall fail if:

21344 [EAGAIN] The requested asynchronous I/O operation was not queued due to system
21345 resource limitations.

21346 Each of the following conditions may be detected synchronously at the time of the call to
21347 *aio_read*( ), or asynchronously. If any of the conditions below are detected synchronously, the

aio_read( ) function shall return –1 and set *errno* to the corresponding value. If any of the conditions below are detected asynchronously, the return status of the asynchronous operation is set to –1, and the error status of the asynchronous operation is set to the corresponding value.

[EBADF] The *aiocbp–>aio_fildes* argument is not a valid file descriptor open for reading.

[EINVAL] The file offset value implied by *aiocbp–>aio_offset* would be invalid,

PIO *aiocbp–>aio_reqprio* is not a valid value, or *aiocbp–>aio_nbytes* is an invalid value.

In the case that the *aio_read*( ) successfully queues the I/O operation but the operation is subsequently canceled or encounters an error, the return status of the asynchronous operation is one of the values normally returned by the *read*( ) function call. In addition, the error status of the asynchronous operation is set to one of the error statuses normally set by the *read*( ) function call, or one of the following values:

[EBADF] The *aiocbp–>aio_fildes* argument is not a valid file descriptor open for reading.

[ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit *aio_cancel*( ) request.

[EINVAL] The file offset value implied by *aiocbp–>aio_offset* would be invalid.

The following condition may be detected synchronously or asynchronously:

[EOVERFLOW] The file is a regular file, *aiobcp–>aio_nbytes* is greater than 0, and the starting offset in *aiobcp–>aio_offset* is before the end-of-file and is at or beyond the offset maximum in the open file description associated with *aiocbp–>aio_fildes*.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_cancel( ), *aio_error*( ), *lio_listio*( ), *aio_return*( ), *aio_write*( ), *close*( ), *exec*, *exit*( ), *fork*( ), *lseek*( ), *read*( )

XBD **<aio.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Large File Summit extensions are added.

Issue 6

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The APPLICATION USAGE section is added.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files.
- In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support large files.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/12 is applied, rewording the DESCRIPTION when prioritized I/O is supported to account for threads, and removing the words ``to the calling process'' in the RETURN VALUE section.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/13 is applied, updating the [EINVAL] error, so that detection of an [EINVAL] error for an invalid value of *aiocbp*->*aio_reqprio* is only required if the Prioritized Input and Output option is supported.

Issue 7

Austin Group Interpretation 1003.1-2001 #082 is applied.

The *aio_read*( ) function is moved from the Asynchronous Input and Output option to the Base.

21402 **NAME**

21403 aio_return — retrieve return status of an asynchronous I/O operation

21404 **SYNOPSIS**

21405 `#include <aio.h>`

21406 `ssize_t aio_return(struct aiocb *aiocbp);`

21407 **DESCRIPTION**

21408 The *aio_return*( ) function shall return the return status associated with the **aiocb** structure
21409 referenced by the *aiocbp* argument. The return status for an asynchronous I/O operation is the
21410 value that would be returned by the corresponding *read*( ), *write*( ), or *fsync*( ) function call. If the
21411 error status for the operation is equal to [EINPROGRESS], then the return status for the
21412 operation is undefined. The *aio_return*( ) function may be called exactly once to retrieve the
21413 return status of a given asynchronous operation; thereafter, if the same **aiocb** structure is used in
21414 a call to *aio_return*( ) or *aio_error*( ), an error may be returned. When the **aiocb** structure referred
21415 to by *aiocbp* is used to submit another asynchronous operation, then *aio_return*( ) may be
21416 successfully used to retrieve the return status of that operation.

21417 **RETURN VALUE**

21418 If the asynchronous I/O operation has completed, then the return status, as described for *read*( ),
21419 *write*( ), and *fsync*( ), shall be returned. If the asynchronous I/O operation has not yet completed,
21420 the results of *aio_return*( ) are undefined.

21421 If the *aio_return*( ) function fails, it shall return –1 and set *errno* to indicate the error.

21422 **ERRORS**

21423 The *aio_return*( ) function may fail if:

21424 [EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose
21425 return status has not yet been retrieved.

21426 **EXAMPLES**

21427 None.

21428 **APPLICATION USAGE**

21429 None.

21430 **RATIONALE**

21431 None.

21432 **FUTURE DIRECTIONS**

21433 None.

21434 **SEE ALSO**

21435 *aio_cancel*( ), *aio_error*( ), *aio_fsync*( ), *aio_read*( ), *aio_write*( ), *close*( ), *exec*, *exit*( ), *fork*( ), *lio_listio*( ),
21436 *lseek*( ), *read*( )

21437 XBD **<aio.h>**

21438 **CHANGE HISTORY**

21439 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

21440 **Issue 6**

21441 The [ENOSYS] error condition has been removed as stubs need not be provided if an
21442 implementation does not support the Asynchronous Input and Output option.

21443 The APPLICATION USAGE section is added.

21444 The [EINVAL] error condition is made optional. This is for consistency with the DESCRIPTION.

Issue 7

SD5-XSH-ERN-148 is applied.

The *aio_return*( ) function is moved from the Asynchronous Input and Output option to the Base.

21448 **NAME**

21449 aio_suspend — wait for an asynchronous I/O request

21450 **SYNOPSIS**

21451 ```
#include <aio.h>
```

21452 ```
int aio_suspend(const struct aiocb *const list[], int nent,
```

21453 ```
    const struct timespec *timeout);
```

21454 **DESCRIPTION**

21455 The *aio_suspend*( ) function shall suspend the calling thread until at least one of the asynchronous
21456 I/O operations referenced by the *list* argument has completed, until a signal interrupts the
21457 function, or, if *timeout* is not NULL, until the time interval specified by *timeout* has passed. If any
21458 of the **aiocb** structures in the list correspond to completed asynchronous I/O operations (that is,
21459 the error status for the operation is not equal to [EINPROGRESS]) at the time of the call, the
21460 function shall return without suspending the calling thread. The *list* argument is an array of
21461 pointers to asynchronous I/O control blocks. The *nent* argument indicates the number of
21462 elements in the array. Each **aiocb** structure pointed to has been used in initiating an
21463 asynchronous I/O request via *aio_read*( ), *aio_write*( ), or *lio_listio*( ). This array may contain null
21464 pointers, which are ignored. If this array contains pointers that refer to **aiocb** structures that have
21465 not been used in submitting asynchronous I/O, the effect is undefined.

21466 If the time interval indicated in the **timespec** structure pointed to by *timeout* passes before any of
21467 the I/O operations referenced by *list* are completed, then *aio_suspend*( ) shall return with an error.
21468 The clock that is used to measure this time interval shall be the CLOCK_MONOTONIC clock.

21469 **RETURN VALUE**

21470 If the *aio_suspend*( ) function returns after one or more asynchronous I/O operations have
21471 completed, the function shall return zero. Otherwise, the function shall return a value of −1 and
21472 set *errno* to indicate the error.

21473 The application may determine which asynchronous I/O completed by scanning the associated
21474 error and return status using *aio_error*( ) and *aio_return*( ), respectively.

21475 **ERRORS**

21476 The *aio_suspend*( ) function shall fail if:

21477 [EAGAIN] No asynchronous I/O indicated in the list referenced by *list* completed in the
21478 time interval indicated by *timeout*.

21479 [EINTR] A signal interrupted the *aio_suspend*( ) function. Note that, since each
21480 asynchronous I/O operation may possibly provoke a signal when it
21481 completes, this error return may be caused by the completion of one (or more)
21482 of the very I/O operations being awaited.

21483 **EXAMPLES**

21484 None.

21485 **APPLICATION USAGE**

21486 None.

21487 **RATIONALE**

21488 None.

21489 **FUTURE DIRECTIONS**

21490 None.

  
```

21491 **SEE ALSO**

21492 *aio_read*( ), *aio_write*( ), *lio_listio*( )

21493 XBD **<aio.h>**

21494 **CHANGE HISTORY**

21495 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

21496 **Issue 6**

21497 The [ENOSYS] error condition has been removed as stubs need not be provided if an
21498 implementation does not support the Asynchronous Input and Output option.

21499 The APPLICATION USAGE section is added.

21500 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the
21501 CLOCK_MONOTONIC clock, if supported, is used.

21502 **Issue 7**

21503 The *aio_suspend*( ) function is moved from the Asynchronous Input and Output option to the
21504 Base.

21505 **Issue 8**

21506 Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.

NAME

aio_write — asynchronous write to a file

SYNOPSIS

```
#include <aio.h>

int aio_write(struct aiocb *aiocbp);
```

DESCRIPTION

The *aio_write*( ) function shall write *aiocbp*−>*aio_nbytes* to the file associated with *aiocbp*−>*aio_fildes* from the buffer pointed to by *aiocbp*−>*aio_buf*. The function shall return when the write request has been initiated or, at a minimum, queued to the file or device.

PIO If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted at a priority equal to a base scheduling priority minus *aiocbp*−>*aio_reqprio*. If Thread Execution Scheduling is not supported, then the base scheduling priority is that of the calling process; PIO TPS otherwise, the base scheduling priority is that of the calling thread.

The *aiocbp* argument may be used as an argument to *aio_error*( ) and *aio_return*( ) in order to determine the error status and return status, respectively, of the asynchronous operation while it is proceeding.

The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp*−>*aio_buf* or the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O completion, then the behavior is undefined.

If O_APPEND is not set for the file descriptor *aio_fildes*, then the requested operation shall take place at the absolute position in the file as given by *aio_offset*, as if *lseek*( ) were called immediately prior to the operation with an *offset* equal to *aio_offset* and a *whence* equal to SEEK_SET. If O_APPEND is set for the file descriptor, or if *aio_fildes* is associated with a device that is incapable of seeking, write operations append to the file in the same order as the calls were made, except under circumstances described in Section 2.8.2. After a successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file is unspecified.

The *aio_sigevent* member specifies the notification which occurs when the request is completed.

The *aiocbp*−>*aio_lio_opcode* field shall be ignored by *aio_write*( ).

Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

SIO If synchronized I/O is enabled on the file associated with *aiocbp*−>*aio_fildes*, the behavior of this function shall be according to the definitions of synchronized I/O data integrity completion, and synchronized I/O file integrity completion.

For any system action that changes the process memory space while an asynchronous I/O is outstanding to the address range being changed, the result of that action is undefined.

For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *aiocbp*−>*aio_fildes*.

If the request would cause the file size to exceed the soft file size limit for the process and there XSI is no room for any bytes to be written, the request shall fail and the implementation shall generate a SIGXFSZ signal for the thread.

RETURN VALUE

The *aio_write*( ) function shall return the value zero if the I/O operation is successfully queued; otherwise, the function shall return the value −1 and set *errno* to indicate the error.

21549 **ERRORS**

21550 The *aio_write*( ) function shall fail if:

21551 [EAGAIN] The requested asynchronous I/O operation was not queued due to system
21552 resource limitations.

21553 Each of the following conditions may be detected synchronously at the time of the call to
21554 *aio_write*( ), or asynchronously. If any of the conditions below are detected synchronously, the
21555 *aio_write*( ) function shall return –1 and set *errno* to the corresponding value. If any of the
21556 conditions below are detected asynchronously, the return status of the asynchronous operation
21557 shall be set to –1, and the error status of the asynchronous operation is set to the corresponding
21558 value.

21559 [EBADF] The *aiocbp*->*aio_fildes* argument is not a valid file descriptor open for writing.

21560 [EINVAL] The file offset value implied by *aiocbp*->*aio_offset* would be invalid,
21561 PIO *aiocbp*->*aio_reqprio* is not a valid value, or *aiocbp*->*aio_nbytes* is an invalid
21562 value.

21563 In the case that the *aio_write*( ) successfully queues the I/O operation, the return status of the
21564 asynchronous operation shall be one of the values normally returned by the *write*( ) function call.
21565 If the operation is successfully queued but is subsequently canceled or encounters an error, the
21566 error status for the asynchronous operation contains one of the values normally set by the
21567 *write*( ) function call, or one of the following:

21568 [EBADF] The *aiocbp*->*aio_fildes* argument is not a valid file descriptor open for writing.

21569 [EINVAL] The file offset value implied by *aiocbp*->*aio_offset* would be invalid.

21570 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit
21571 *aio_cancel*( ) request.

21572 The following condition may be detected synchronously or asynchronously:

21573 [EFBIG] The file is a regular file, *aiobcp*->*aio_nbytes* is greater than 0, and the starting
21574 position is greater than or equal to the offset maximum in the open file
21575 description associated with *aiocbp*->*aio_fildes*.

21576 [EFBIG] The file is a regular file, *aiobcp*->*aio_nbytes* is greater than 0, and there is no
21577 room for any bytes to be written at the starting position without exceeding the
21578 XSI file size limit for the process. A SIGXFSZ signal shall also be generated for
21579 the thread.

21580 [EFBIG] The file is a regular file, *aiobcp*->*aio_nbytes* is greater than 0, and there is no
21581 room for any bytes to be written at the starting position without exceeding the
21582 implementation-defined maximum file size.

21583 **EXAMPLES**

21584 None.

21585 **APPLICATION USAGE**

21586 None.

21587 **RATIONALE**

21588 None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.8.2 (on page 528), *aio_cancel*( ), *aio_error*( ), *aio_read*( ), *aio_return*( ), *close*( ), *exec*, *exit*( ), *fork*( ), *lio_listio*( ), *lseek*( ), *write*( )

XBD **<aio.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Large File Summit extensions are added.

Issue 6

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The APPLICATION USAGE section is added.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs past the offset maximum established in the open file description associated with *aiocbp->aio_fildes*.
- The [EFBIG] error is added as part of the large file support extensions.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/14 is applied, rewording the DESCRIPTION when prioritized I/O is supported to account for threads, and removing the words ``to the calling process'' in the RETURN VALUE section.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/15 is applied, updating the [EINVAL] error, so that detection of an [EINVAL] error for an invalid value of *aiocbp->aio_reqprio* is only required if the Prioritized Input and Output option is supported.

Issue 7

Austin Group Interpretation 1003.1-2001 #082 is applied.

The *aio_write*( ) function is moved from the Asynchronous Input and Output option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0028 [317] is applied.

Issue 8

Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error relating to the file size limit for the process.

21622 **NAME**

21623 alarm — schedule an alarm signal

21624 **SYNOPSIS**

21625 `#include <unistd.h>`

21626 `unsigned alarm(unsigned seconds);`

21627 **DESCRIPTION**

21628 The *alarm*( ) function shall cause the system to generate a SIGALRM signal for the process after
21629 the number of realtime seconds specified by *seconds* have elapsed. Processor scheduling delays
21630 may prevent the process from handling the signal as soon as it is generated.

21631 If *seconds* is 0, a pending alarm request, if any, is canceled.

21632 Alarm requests are not stacked; only one SIGALRM generation can be scheduled in this manner.
21633 If the SIGALRM signal has not yet been generated, the call shall result in rescheduling the time
21634 at which the SIGALRM signal is generated.

21635 **RETURN VALUE**

21636 If there is a previous *alarm*( ) request with time remaining, *alarm*( ) shall return a non-zero value
21637 that is the number of seconds until the previous request would have generated a SIGALRM
21638 signal. Otherwise, *alarm*( ) shall return 0.

21639 **ERRORS**

21640 The *alarm*( ) function is always successful, and no return value is reserved to indicate an error.

21641 **EXAMPLES**

21642 None.

21643 **APPLICATION USAGE**

21644 The *fork*( ) function clears pending alarms in the child process. A new process image created by
21645 one of the *exec* functions inherits the time left to an alarm signal in the image of the old process.

21646 Application developers should note that the type of the argument *seconds* and the return value of
21647 *alarm*( ) is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces
21648 Application cannot pass a value greater than the minimum guaranteed value for {UINT_MAX},
21649 which the ISO C standard sets as 65 535, and any application passing a larger value is restricting
21650 its portability. A different type was considered, but historical implementations, including those
21651 with a 16-bit **int** type, consistently use either **unsigned** or **int**.

21652 Application developers should be aware of possible interactions when the same process uses
21653 both the *alarm*( ) and *sleep*( ) functions.

21654 **RATIONALE**

21655 Many historical implementations (including Version 7 and System V) allow an alarm to occur up
21656 to a second early. Other implementations allow alarms up to half a second or one clock tick
21657 early or do not allow them to occur early at all. The latter is considered most appropriate, since it
21658 gives the most predictable behavior, especially since the signal can always be delayed for an
21659 indefinite amount of time due to scheduling. Applications can thus choose the *seconds* argument
21660 as the minimum amount of time they wish to have elapse before the signal.

21661 The term ``realtime'' here and elsewhere (*sleep*( ), *times*( )) is intended to mean ``wall clock'' time
21662 as common English usage, and has nothing to do with ``realtime operating systems''. It is in
21663 contrast to *virtual time*, which could be misinterpreted if just *time* were used.

21664 In some implementations, including 4.3 BSD, very large values of the *seconds* argument are
21665 silently rounded down to an implementation-specific maximum value. This maximum is large
21666 enough (to the order of several months) that the effect is not noticeable.

There were two possible choices for alarm generation in multi-threaded applications: generation for the calling thread or generation for the process. The first option would not have been particularly useful since the alarm state is maintained on a per-process basis and the alarm that is established by the last invocation of *alarm*( ) is the only one that would be active.

Furthermore, allowing generation of an asynchronous signal for a thread would have introduced an exception to the overall signal model. This requires a compelling reason in order to be justified.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *exec*, *fork*( ), *pause*( ), *sigaction*( ), *sleep*( ), *timer_create*( )

XBD **<signal.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to indicate that interactions with the *setitimer*( ), *ualarm*( ), and *usleep*( ) functions are unspecified.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/16 is applied, replacing ``an implementation-defined maximum value'' with ``an implementation-specific maximum value'' in the RATIONALE.

Issue 8

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

21691 **NAME**

21692 aligned_alloc — allocate memory with a specified alignment

21693 **SYNOPSIS**

21694
```
#include <stdlib.h>
```

21695
```
void *aligned_alloc(size_t alignment, size_t size);
```

21696 **DESCRIPTION**

21697 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21698 conflict between the requirements described here and the ISO C standard is unintentional. This
21699 volume of POSIX.1-2024 defers to the ISO C standard.

21700 The *aligned_alloc*( ) function shall allocate unused space for an object whose alignment is
21701 specified by *alignment*, whose size in bytes is specified by *size*, and whose value is indeterminate.

21702 The order and contiguity of storage allocated by successive calls to *aligned_alloc*( ) is unspecified.
21703 Each such allocation shall yield a pointer to an object disjoint from any other object. The pointer
21704 returned shall point to the start (lowest byte address) of the allocated space. If the value of
21705 *alignment* is not a valid alignment supported by the implementation, a null pointer shall be
21706 returned. If the space cannot be allocated, a null pointer shall be returned. If the size of the space
21707 requested is 0, the behavior is implementation-defined: either a null pointer shall be returned to
21708 indicate an error, or the behavior shall be as if the size were some non-zero value, except that the
21709 behavior is undefined if the returned pointer is used to access an object.

21710 For purposes of determining the existence of a data race, *aligned_alloc*( ) shall behave as though it
21711 accessed only memory locations accessible through its arguments and not other static duration
21712 storage. The function may, however, visibly modify the storage that it allocates. Calls to
21713 ADV *aligned_alloc*( ), *calloc*( ), *free*( ), *malloc*( ), *posix_memalign*( ),
21714 CX *reallocarray*( ), and *realloc*( ) that allocate or deallocate a particular region of memory shall occur
21715 in a single total order (see Section 4.15.1, on page 100), and each such deallocation call shall
21716 synchronize with the next allocation (if any) in this order.

21717 **RETURN VALUE**

21718 Upon successful completion, *aligned_alloc*( ) shall return a pointer to the allocated space; if *size* is
21719 0, the application shall ensure that the pointer is not used to access an object.

21720 CX Otherwise, it shall return a null pointer and set *errno* to indicate the error.

21721 **ERRORS**

21722 The *aligned_alloc*( ) function shall fail if:

21723 CX [EINVAL] The value of *alignment* is not a valid alignment supported by the
21724 implementation.

21725 CX [ENOMEM] Insufficient storage space is available.

21726 The *aligned_alloc*( ) function may fail if:

21727 CX [EINVAL] *size* is 0 and the implementation does not support 0 sized allocations.

21728 **EXAMPLES**

21729 None.

21730 **APPLICATION USAGE**

21731 None.

21732 **RATIONALE**

21733 See the RATIONALE for *malloc( )*.

21734 **FUTURE DIRECTIONS**

21735 None.

21736 **SEE ALSO**

21737 *calloc( )*, *free( )*, *getrlimit( )*, *malloc( )*, *posix_memalign( )*, *realloc( )*

21738 XBD **<stdlib.h>**

21739 **CHANGE HISTORY**

21740 First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

alphasort, scandir — scan a directory

SYNOPSIS

```
#include <dirent.h>

int alphasort(const struct dirent **d1, const struct dirent **d2);
int scandir(const char *dir, struct dirent ***namelist,
    int (*sel)(const struct dirent *),
    int (*compar)(const struct dirent **, const struct dirent **));
```

DESCRIPTION

The *alphasort*( ) function can be used as the comparison function for the *scandir*( ) function to sort the directory entries, *d1* and *d2*, into alphabetical order. Sorting happens as if by calling the *strcoll*( ) function on the *d_name* element of the **dirent** structures passed as the two parameters. If the *strcoll*( ) function fails, the return value of *alphasort*( ) is unspecified.

The *alphasort*( ) function shall not change the setting of *errno* if successful. Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *alphasort*( ), then check *errno*.

The *scandir*( ) function shall scan the directory *dir*, calling the function referenced by *sel* on each directory entry. Entries for which the function referenced by *sel* returns non-zero shall be stored in strings allocated as if by a call to *malloc*( ), and sorted as if by a call to *qsort*( ) with the comparison function *compar*, except that *compar* need not provide total ordering. The strings are collected in array *namelist* which shall be allocated as if by a call to *malloc*( ). If *sel* is a null pointer, all entries shall be selected. If the comparison function *compar* does not provide total ordering, the order in which the directory entries are stored is unspecified.

RETURN VALUE

Upon successful completion, the *alphasort*( ) function shall return an integer greater than, equal to, or less than 0, according to whether the name of the directory entry pointed to by *d1* is lexically greater than, equal to, or less than the directory pointed to by *d2* when both are interpreted as appropriate to the current locale. There is no return value reserved to indicate an error.

Upon successful completion, the *scandir*( ) function shall return the number of entries in the array and a pointer to the array through the parameter *namelist*. Otherwise, the *scandir*( ) function shall return –1.

ERRORS

The *scandir*( ) function shall fail if:

[EACCES] Search permission is denied for the component of the path prefix of *dir* or read permission is denied for *dir*.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *dir* argument.

[ENAMETOOLONG] The length of a component of a pathname is longer than {NAME_MAX}.

[ENOENT] A component of *dir* does not name an existing directory or *dir* is an empty string.

[ENOMEM] Insufficient storage space is available.

[ENOTDIR] A component of *dir* names an existing file that is neither a directory nor a symbolic link to a directory.

[EOVERFLOW] One of the values to be returned or passed to a callback function cannot be represented correctly.

The *scandir*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *dir* argument.

[EMFILE] All file descriptors available to the process are currently open.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

[ENFILE] Too many files are currently open in the system.

EXAMPLES

An example to print the files in the current directory:

```
#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>
...
struct dirent **namelist;
int i,n;

    n = scandir(".", &namelist, 0, alphasort);
    if (n < 0)
        perror("scandir");
    else {
        for (i = 0; i < n; i++) {
            printf("%s\n", namelist[i]->d_name);
            free(namelist[i]);
            }
        }
    free(namelist);
...
```

APPLICATION USAGE

If *dir* contains filenames that do not form character strings, or which contain characters outside the domain of the collating sequence of the current locale, the *alphasort*( ) function need not provide a total ordering. This condition is not possible if all filenames within the directory consist only of characters from the portable filename character set.

The *scandir*( ) function may allocate dynamic storage during its operation. If *scandir*( ) is forcibly terminated, such as by *longjmp*( ) or *siglongjmp*( ) being executed by the function pointed to by *sel* or *compar*, or by an interrupt routine, *scandir*( ) does not have a chance to free that storage, so it remains permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, then wait until *scandir*( ) returns to act on the interrupt.

For functions that allocate memory as if by *malloc*( ), the application should release such memory when it is no longer required by a call to *free*( ). For *scandir*( ), this is *namelist* (including all of the individual strings in *namelist*).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

qsort( ), *strcoll*( )

XBD **<dirent.h>**

CHANGE HISTORY

First released in Issue 7.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0029 [324], XSH/TC1-2008/0030 [404], XSH/TC1-2008/0031 [393], and XSH/TC1-2008/0032 [291] are applied.

NAME

asctime — convert date and time to a string

SYNOPSIS

OB

```
#include <time.h>

char *asctime(const struct tm *timeptr);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *asctime*( ) function shall convert the broken-down time in the structure pointed to by *timeptr* into a string in the form:

```
Sun Sep 16 01:03:52 1973\n\0
```

using the equivalent of the following algorithm:

```
char *asctime(const struct tm *timeptr)
{
    static char wday_name[7][3] = {
        "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
    };
    static char mon_name[12][3] = {
        "Jan", "Feb", "Mar", "Apr", "May", "Jun",
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
    };
    static char result[26];

    sprintf(result, "%.3s %.3s%3d %.2d:%.2d:%.2d %d\n",
        wday_name[timeptr->tm_wday],
        mon_name[timeptr->tm_mon],
        timeptr->tm_mday, timeptr->tm_hour,
        timeptr->tm_min, timeptr->tm_sec,
        1900 + timeptr->tm_year);
    return result;
}
```

If any of the members of the broken-down time contain values that are outside their normal ranges (see XBD **<time.h>**), the behavior of the *asctime*( ) function is undefined. Likewise, if the calculated year exceeds four digits or is less than the year 1000, the behavior is undefined.

The **tm** structure is defined in the **<time.h>** header.

The *asctime*( ), *ctime*( ), *gmtime*( ), and *localtime*( ) functions shall return values in one of two static objects: a broken-down time structure and an array of type **char**. Execution of any of the functions that return a pointer to one of these object types may overwrite the information in any object of the same type pointed to by the value returned from any previous call to any of them.

The *asctime*( ) function need not be thread-safe; however, *asctime*( ) shall avoid data races with all functions other than itself, *ctime*( ), *gmtime*( ), and *localtime*( ).

RETURN VALUE

CX Upon successful completion, *asctime*( ) shall return a pointer to the string. If the function is unsuccessful, it shall return NULL.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

This function is included only for compatibility with older implementations. It has undefined behavior if the resulting string would be too long, so the use of this function should be discouraged. On implementations that do not detect output string length overflow, it is possible to overflow the output buffer in such a way as to cause applications to fail, or possible system security violations. Also, this function does not support localized date and time formats. To avoid these problems, applications should use *strftime*( ) to generate strings from broken-down times.

Values for the broken-down time structure can be obtained by calling *gmtime*( ) or *localtime*( ).

RATIONALE

The standard developers decided to mark the *asctime*( ) function obsolescent even though it is in the ISO C standard due to the possibility of buffer overflow. The ISO C standard also provides the *strftime*( ) function which can be used to avoid these problems.

FUTURE DIRECTIONS

This function may be removed in a future version, but not until after it has been removed from the ISO C standard.

SEE ALSO

clock( ), *ctime*( ), *difftime*( ), *futimens*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( )

XBD **<time.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

The *asctime_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *asctime*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *asctime_r*( ) function is marked as part of the Thread-Safe Functions option.

Extensions beyond the ISO C standard are marked.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

The DESCRIPTION of *asctime_r*( ) is updated to describe the format of the string returned.

The **restrict** keyword is added to the *asctime_r*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/17 is applied, adding the CX extension in the RETURN VALUE section requiring that if the *asctime*( ) function is unsuccessful it returns NULL.

Issue 7

Austin Group Interpretation 1003.1-2001 #053 is applied, marking these functions obsolescent.

Austin Group Interpretation 1003.1-2001 #156 is applied.

The *asctime_r*( ) function is moved from the Thread-Safe Functions option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0033 [86,429] is applied.

Issue 8

Austin Group Defect 469 is applied, clarifying the conditions under which the behavior of *asctime*( ) is undefined.

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018 standard.

Austin Group Defect 1330 is applied, changing the FUTURE DIRECTIONS section.

Austin Group Defect 1376 is applied, removing CX shading from some text derived from the ISO C standard and updating it to match the ISO C standard.

Austin Group Defect 1410 is applied, removing the *asctime_r*( ) function.

NAME

asin, asinf, asinl — arc sine function

SYNOPSIS

```
#include <math.h>

double asin(double x);
float asinf(float x);
long double asinl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the principal value of the arc sine of their argument *x*. The value of *x* should be in the range [–1,1].

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the arc sine of *x*, in the range $[-\pi/2,\pi/2]$ radians.

MX For finite values of *x* not in the range [–1,1], a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is ±Inf, a domain error shall occur, and a NaN shall be returned.

MXX If *x* is subnormal, *x* should be returned.

MX If *x* is subnormal, a range error may occur and, if *x* is not returned, *asin*( ), *asinf*( ), and *asinl*( ) shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

ERRORS

These functions shall fail if:

MX Domain Error — The *x* argument is finite and is not in the range [–1,1], or is ±Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

MX Range Error — The value of *x* is subnormal.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( ), *sin*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *asinf*( ) and *asinl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0034 [320] and XSH/TC1-2008/0035 [68] are applied.

Issue 8

Austin Group Defect 1382 is applied, rearranging the text describing the behavior when *x* is subnormal to avoid the need for two shading changes.

NAME

asinh, asinhf, asinhl — inverse hyperbolic sine functions

SYNOPSIS

```
#include <math.h>

double asinh(double x);
float asinhf(float x);
long double asinhl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the inverse hyperbolic sine of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the inverse hyperbolic sine of their argument.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, or ±Inf, *x* shall be returned.

MXX If *x* is subnormal, *x* should be returned.

MX If *x* is subnormal, a range error may occur and, if *x* is not returned, *asinh*( ), *asinhf*( ), and *asinhl*( ) shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

ERRORS

These functions may fail if:

MX Range Error — The value of *x* is subnormal.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

22052 **SEE ALSO**

22053 *feclearexcept*( ), *fetestexcept*( ), *sinh*( )

22054 XBD Section 4.23 (on page 109), **<math.h>**

22055 **CHANGE HISTORY**

22056 First released in Issue 4, Version 2.

22057 **Issue 5**

22058 Moved from X/OPEN UNIX extension to BASE.

22059 **Issue 6**

22060 The *asinh*( ) function is no longer marked as an extension.

22061 The *asinhf*( ) and *asinhl*( ) functions are added for alignment with the ISO/IEC 9899:1999
22062 standard.

22063 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
22064 revised to align with the ISO/IEC 9899:1999 standard.

22065 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
22066 marked.

22067 **Issue 7**

22068 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0036 [68] is applied.

22069 **Issue 8**

22070 Austin Group Defect 1382 is applied, rearranging the text describing the behavior when *x* is
22071 subnormal to avoid the need for two shading changes.

22072 **NAME**

22073 asinl — arc sine function

22074 **SYNOPSIS**

22075 `#include <math.h>`

22076 `long double asinl(long double x);`

22077 **DESCRIPTION**

22078 Refer to *asin*( ).

22079 **NAME**

22080 asprintf — print formatted output

22081 **SYNOPSIS**

22082 `#include <stdio.h>`

22083 `int asprintf(char **restrict ptr, const char *restrict format, ...);`

22084 **DESCRIPTION**

22085 Refer to *fprintf( )*.

NAME

assert — insert program diagnostics

SYNOPSIS

```
#include <assert.h>

void assert(scalar expression);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *assert*( ) macro shall insert diagnostics into programs; it shall expand to a **void** expression. When it is executed, if *expression* (which shall have a **scalar** type) is false (that is, compares equal to 0), *assert*( ) shall write information about the particular call that failed on *stderr* and shall call *abort*( ).

The information written about the call that failed shall include the text of the argument, the name of the source file, the source file line number, and the name of the enclosing function; the latter are, respectively, the values of the preprocessing macros __FILE__ and __LINE__ and of the identifier __func__.

Forcing a definition of the name NDEBUG, either from the compiler command line or with the preprocessor control statement **#define** NDEBUG ahead of the **#include <assert.h>** statement, shall stop assertions from being compiled into the program.

RETURN VALUE

The *assert*( ) macro shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

abort( ), *stdin*

XBD **<assert.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The prototype for the *expression* argument to *assert*( ) is changed from **int** to **scalar** for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION of *assert*( ) is updated for alignment with the ISO/IEC 9899: 1999 standard.

22127 **NAME**

22128 at_quick_exit — register a function to to be called from *quick_exit*( )

22129 **SYNOPSIS**

22130 `#include <stdlib.h>`

22131 `int at_quick_exit(void (*func)(void));`

22132 **DESCRIPTION**

22133 CX The functionality described on this reference page is aligned with the ISO C standard. Any
22134 conflict between the requirements described here and the ISO C standard is unintentional. This
22135 volume of POSIX.1-2024 defers to the ISO C standard.

22136 The *at_quick_exit*( ) function shall register the function pointed to by *func*, to be called without
22137 arguments should *quick_exit*( ) be called. It is unspecified whether a call to the *at_quick_exit*( )
22138 function that does not happen before the *quick_exit*( ) function is called will succeed.

22139 At least 32 functions can be registered with *at_quick_exit*( ).

22140 **RETURN VALUE**

22141 Upon successful completion, *at_quick_exit*( ) shall return 0; otherwise, it shall return a non-zero
22142 value.

22143 **ERRORS**

22144 No errors are defined.

22145 **EXAMPLES**

22146 None.

22147 **APPLICATION USAGE**

22148 The *at_quick_exit*( ) function registrations are distinct from the *atexit*( ) registrations, so
22149 applications might need to call both registration functions with the same argument.

22150 The functions registered by a call to *at_quick_exit*( ) must return to ensure that all registered
22151 functions are called.

22152 The application should call *sysconf*( ) to obtain the value of {ATEXIT_MAX}, the number of
22153 functions that can be registered. There is no way for an application to tell how many functions
22154 have already been registered with *at_quick_exit*( ).

22155 Since the behavior is undefined if the *quick_exit*( ) function is called more than once, portable
22156 applications calling *at_quick_exit*( ) must ensure that the *quick_exit*( ) function is not called when
22157 the functions registered by the *at_quick_exit*( ) function are called.

22158 If a function registered by the *at_quick_exit*( ) function is called and a portable application needs
22159 to stop further *quick_exit*( ) processing, it must call the *_exit*( ) function or the *_Exit*( ) function or
22160 one of the functions which cause abnormal process termination.

22161 **RATIONALE**

22162 None.

22163 **FUTURE DIRECTIONS**

22164 None.

22165 **SEE ALSO**

22166 *atexit*( ), *exec*, *exit*( ), *quick_exit*( ), *sysconf*( )

22167 XBD **<stdlib.h>**

22168 **CHANGE HISTORY**

22169 First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

atan, atanf, atanl — arc tangent function

SYNOPSIS

```
#include <math.h>

double atan(double x);
float atanf(float x);
long double atanl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the principal value of the arc tangent of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the arc tangent of *x* in the range $[-\pi/2,\pi/2]$ radians.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is ±Inf, $\pm\pi/2$ shall be returned.

MXX If *x* is subnormal, *x* should be returned.

MX If *x* is subnormal, a range error may occur and, if *x* is not returned, *atan*( ), *atanf*( ), and *atanl*( ) shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

ERRORS

These functions may fail if:

MX Range Error — The value of *x* is subnormal.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

atan2( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *tan*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *atanf*( ) and *atanl*( ) functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0037 [68] is applied.

Issue 8

Austin Group Defect 1382 is applied, rearranging the text describing the behavior when *x* is subnormal to avoid the need for two shading changes.

NAME

atan2, atan2f, atan2l — arc tangent functions

SYNOPSIS

```
#include <math.h>

double atan2(double y, double x);
float atan2f(float y, float x);
long double atan2l(long double y, long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the principal value of the arc tangent of y/x, using the signs of both arguments to determine the quadrant of the return value.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the arc tangent of y/x in the range $[-\pi,\pi]$ radians.

If y is ±0 and x is < 0, $\pm\pi$ shall be returned.

If y is ±0 and x is > 0, ±0 shall be returned.

If y is < 0 and x is ±0, $-\pi/2$ shall be returned.

If y is > 0 and x is ±0, $\pi/2$ shall be returned.

If x is 0, a pole error shall not occur.

MX If either x or y is NaN, a NaN shall be returned.

If the correct value would cause underflow, a range error may occur, and *atan*( ), *atan2f*( ), and *atan2l*( ) shall return an implementation-defined value no greater in magnitude than DBL_MIN, MXX FLT_MIN, and LDBL_MIN, respectively. If the IEC 60559 Floating-Point option is supported, y/x should be returned.

MX If y is ±0 and x is –0, $\pm\pi$ shall be returned.

If y is ±0 and x is +0, ±0 shall be returned.

For finite values of $\pm y > 0$, if x is –Inf, $\pm\pi$ shall be returned.

For finite values of $\pm y > 0$, if x is +Inf, ±0 shall be returned.

For finite values of x, if y is ±Inf, $\pm\pi/2$ shall be returned.

If y is ±Inf and x is –Inf, $\pm 3\pi/4$ shall be returned.

If y is ±Inf and x is +Inf, $\pm\pi/4$ shall be returned.

If both arguments are 0, a domain error shall not occur.

ERRORS

These functions may fail if:

MX

Range Error	The result underflows.
	If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

Converting Cartesian to Polar Coordinates System

The function below uses *atan2*( ) to convert a 2d vector expressed in cartesian coordinates (*x*,*y*) to the polar coordinates (*rho*,*theta*). There are other ways to compute the angle *theta*, using *asin*( ) *acos*( ), or *atan*( ). However, *atan2*( ) presents here two advantages:

- The angle's quadrant is automatically determined.
- The singular cases (0,*y*) are taken into account.

Finally, this example uses *hypot*( ) rather than *sqrt*( ) since it is better for special cases; see *hypot*( ) for more information.

```
#include <math.h>

void
cartesian_to_polar(const double x, const double y,
                   double *rho, double *theta
    )
{
    *rho   = hypot (x,y); /* better than sqrt(x*x+y*y) */
    *theta = atan2 (y,x);
}
```

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

acos( ), *asin*( ), *atan*( ), *feclearexcept*( ), *fetestexcept*( ), *hypot*( ), *isnan*( ), *sqrt*( ), *tan*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *atan2f*( ) and *atan2l*( ) functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard, and the IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/18 is applied, adding to the EXAMPLES section.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0038 [68,428] is applied.

Issue 8

Austin Group Defect 1178 is applied, removing MX shading from a paragraph in the RETURN VALUE section and joining it with the following paragraph.

22322 **NAME**

22323 atanf — arc tangent function

22324 **SYNOPSIS**

22325 `#include <math.h>`

22326 `float atanf(float x);`

22327 **DESCRIPTION**

22328 Refer to *atan*( ).

NAME

atanh, atanhf, atanhl — inverse hyperbolic tangent functions

SYNOPSIS

```
#include <math.h>

double atanh(double x);
float atanhf(float x);
long double atanhl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the inverse hyperbolic tangent of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the inverse hyperbolic tangent of their argument.

If *x* is ±1, a pole error shall occur, and *atanh*( ), *atanhf*( ), and *atanhl*( ) shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively, with the same sign as the correct value of the function.

MX For finite $|x|>1$, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is ±Inf, a domain error shall occur, and a NaN shall be returned.

MXX If *x* is subnormal, *x* should be returned.

MX If *x* is subnormal, a range error may occur and, if *x* is not returned, *atanh*( ), *atanhf*( ), and *atanhl*( ) shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

ERRORS

These functions shall fail if:

MX Domain Error — The *x* argument is finite and not in the range [–1,1], or is ±Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error — The *x* argument is ±1.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

These functions may fail if:

MX Range Error — The value of *x* is subnormal.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *tanh*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The *atanh*( ) function is no longer marked as an extension.

The *atanhf*( ) and *atanhl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0039 [320] and XSH/TC1-2008/0040 [680] are applied.

Issue 8

Austin Group Defect 1382 is applied, rearranging the text describing the behavior when *x* is subnormal to avoid the need for two shading changes.

22408 **NAME**

22409 atanl — arc tangent function

22410 **SYNOPSIS**

22411 `#include <math.h>`

22412 `long double atanl(long double x);`

22413 **DESCRIPTION**

22414 Refer to *atan( )*.

22415 **NAME**

22416 atexit — register a function to be called from *exit*( ) or after return from *main*( )

22417 **SYNOPSIS**

22418 ```
#include <stdlib.h>
```

22419 ```
int atexit(void (*func)(void));
```

22420 **DESCRIPTION**

22421 CX The functionality described on this reference page is aligned with the ISO C standard. Any
22422 conflict between the requirements described here and the ISO C standard is unintentional. This
22423 volume of POSIX.1-2024 defers to the ISO C standard.

22424 The *atexit*( ) function shall register the function pointed to by *func*, to be called without
22425 arguments from *exit*( ), or after return from the initial call to *main*( ), or on the last thread
22426 termination. If the *exit*( ) function is called, it is unspecified whether a call to the *atexit*( ) function
22427 that does not happen before *exit*( ) is called will succeed.

22428 At least 32 functions can be registered with *atexit*( ).

22429 **RETURN VALUE**

22430 Upon successful completion, *atexit*( ) shall return 0; otherwise, it shall return a non-zero value.

22431 **ERRORS**

22432 No errors are defined.

22433 **EXAMPLES**

22434 None.

22435 **APPLICATION USAGE**

22436 The *atexit*( ) function registrations are distinct from the *at_quick_exit*( ) registrations, so
22437 applications might need to call both registration functions with the same argument.

22438 The functions registered by a call to *atexit*( ) must return to ensure that all registered functions
22439 are called.

22440 The application should call *sysconf*( ) to obtain the value of {ATEXIT_MAX}, the number of
22441 functions that can be registered. There is no way for an application to tell how many functions
22442 have already been registered with *atexit*( ).

22443 Since the behavior is undefined if the *exit*( ) function is called more than once, portable
22444 applications calling *atexit*( ) must ensure that the *exit*( ) function is not called when the functions
22445 registered by the *atexit*( ) function are called.

22446 If a function registered by the *atexit*( ) function is called and a portable application needs to stop
22447 further *exit*( ) processing, it must call the *_exit*( ) function or the *_Exit*( ) function or one of the
22448 functions which cause abnormal process termination.

22449 **RATIONALE**

22450 None.

22451 **FUTURE DIRECTIONS**

22452 None.

22453 **SEE ALSO**

22454 *at_quick_exit*( ), *exec*, *exit*( ), *sysconf*( )

22455 XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ANSI C standard.

Issue 6

Extensions beyond the ISO C standard are marked.

The DESCRIPTION is updated for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/19 is applied, adding further clarification to the APPLICATION USAGE section.

Issue 8

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899: 2018 standard.

Austin Group Defect 1646 is applied, removing redundant text relating to the *exec* family of functions.

22468 **NAME**

22469 atof — convert a string to a double-precision number

22470 **SYNOPSIS**

22471 ```
#include <stdlib.h>
```

22472 ```
double atof(const char *str);
```

22473 **DESCRIPTION**

22474 CX The functionality described on this reference page is aligned with the ISO C standard. Any
22475 conflict between the requirements described here and the ISO C standard is unintentional. This
22476 volume of POSIX.1-2024 defers to the ISO C standard.

22477 The call *atof*(*str*) shall be equivalent to:

22478 ```
strtod(str,(char **)NULL),
```

22479 except that the handling of errors may differ. If the value cannot be represented, the behavior is
22480 undefined.

22481 **RETURN VALUE**

22482 The *atof*( ) function shall return the converted value if the value can be represented.

22483 **ERRORS**

22484 No errors are defined.

22485 **EXAMPLES**

22486 None.

22487 **APPLICATION USAGE**

22488 The *atof*( ) function is subsumed by *strtod*( ) but is retained because it is used extensively in
22489 existing code. If the number is not known to be in range, *strtod*( ) should be used because *atof*( )
22490 is not required to perform any error checking.

22491 **RATIONALE**

22492 None.

22493 **FUTURE DIRECTIONS**

22494 None.

22495 **SEE ALSO**

22496 *strtod*( )

22497 XBD **<stdlib.h>**

22498 **CHANGE HISTORY**

22499 First released in Issue 1. Derived from Issue 1 of the SVID.

  
```

NAME

atoi — convert a string to an integer

SYNOPSIS

```
#include <stdlib.h>

int atoi(const char *str);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The call *atoi*(*str*) shall be equivalent to:

```
(int) strtol(str, (char **)NULL, 10)
```

except that the handling of errors may differ. If the value cannot be represented, the behavior is undefined.

RETURN VALUE

The *atoi*( ) function shall return the converted value if the value can be represented.

ERRORS

No errors are defined.

EXAMPLES

Converting an Argument

The following example checks for proper usage of the program. If there is an argument and the decimal conversion of this argument (obtained using *atoi*( )) is greater than 0, then the program has a valid number of minutes to wait for an event.

```
#include <stdlib.h>
#include <stdio.h>
...
int minutes_to_event;
...
if (argc < 2 || (minutes_to_event = atoi (argv[1])) <= 0) {
   fprintf(stderr, "Usage: %s minutes\n", argv[0]); exit(1);
}
...
```

APPLICATION USAGE

The *atoi*( ) function is subsumed by *strtol*( ) but is retained because it is used extensively in existing code. If the number is not known to be in range, *strtol*( ) should be used because *atoi*( ) is not required to perform any error checking.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strtol( )

XBD **<stdlib.h>**

22542 **CHANGE HISTORY**

22543 First released in Issue 1. Derived from Issue 1 of the SVID.

22544 **Issue 8**

22545 Austin Group Defect 1541 is applied, changing the EXAMPLES section.

NAME

atol, atoll — convert a string to a long integer

SYNOPSIS

```
#include <stdlib.h>

long atol(const char *nptr);
long long atoll(const char *nptr);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

Except as noted below, the call *atol*(*nptr*) shall be equivalent to:

```
strtol(nptr, (char **)NULL, 10)
```

Except as noted below, the call to *atoll*(*nptr*) shall be equivalent to:

```
strtoll(nptr, (char **)NULL, 10)
```

The handling of errors may differ. If the value cannot be represented, the behavior is undefined.

RETURN VALUE

These functions shall return the converted value if the value can be represented.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

If the number is not known to be in range, *strtol*( ) or *strtoll*( ) should be used because *atol*( ) and *atoll*( ) are not required to perform any error checking.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strtol( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The *atoll*( ) function is added for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

SD5-XSH-ERN-61 is applied, correcting the DESCRIPTION of *atoll*( ).

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0046 [892] is applied.

NAME

atomic_compare_exchange_strong, atomic_compare_exchange_strong_explicit, atomic_compare_exchange_weak, atomic_compare_exchange_weak_explicit — atomically compare and exchange the values of two objects

SYNOPSIS

```
#include <stdatomic.h>

_Bool atomic_compare_exchange_strong(volatile A *object,
    C *expected, C desired);
_Bool atomic_compare_exchange_strong_explicit(volatile A *object,
    C *expected, C desired, memory_order success, memory_order failure);
_Bool atomic_compare_exchange_weak(volatile A *object,
    C *expected, C desired);
_Bool atomic_compare_exchange_weak_explicit(volatile A *object,
    C *expected, C desired, memory_order success, memory_order failure);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

Implementations that define the macro __STDC_NO_ATOMICS__ need not provide the **<stdatomic.h>** header nor support these generic functions.

The *atomic_compare_exchange_strong_explicit*( ) generic function shall atomically compare the contents of the memory pointed to by *object* for equality with that pointed to by *expected*, and if true, shall replace the contents of the memory pointed to by *object* with *desired*, and if false, shall update the contents of the memory pointed to by *expected* with that pointed to by *object*. This operation shall be an atomic read-modify-write operation (see XBD Section 4.15.1, on page 100). If the comparison is true, *memory* shall be affected according to the value of *success*, and if the comparison is false, *memory* shall be affected according to the value of *failure*. The application shall ensure that *failure* is not `memory_order_release` nor `memory_order_acq_rel`, and shall ensure that *failure* is no stronger than *success*.

The *atomic_compare_exchange_strong*( ) generic function shall be equivalent to *atomic_compare_exchange_strong_explicit*( ) called with *success* and *failure* both set to `memory_order_seq_cst`.

The *atomic_compare_exchange_weak_explicit*( ) generic function shall be equivalent to *atomic_compare_exchange_strong_explicit*( ), except that the compare-and-exchange operation may fail spuriously. That is, even when the contents of *memory* referred to by *expected* and *object* are equal, it may return zero and store back to *expected* the same memory contents that were originally there.

The *atomic_compare_exchange_weak*( ) generic function shall be equivalent to *atomic_compare_exchange_weak_explicit*( ) called with *success* and *failure* both set to `memory_order_seq_cst`.

RETURN VALUE

These generic functions shall return the result of the comparison.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

A consequence of spurious failure is that nearly all uses of weak compare-and-exchange will be in a loop. For example:

```
exp = atomic_load(&cur);
do {
    des = function(exp);
} while (!atomic_compare_exchange_weak(&cur, &exp, des));
```

When a compare-and-exchange is in a loop, the weak version will yield better performance on some platforms. When a weak compare-and-exchange would require a loop and a strong one would not, the strong one is preferable.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD Section 4.15.1, **<stdatomic.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

22648 **NAME**

22649 atomic_exchange, atomic_exchange_explicit — atomically exchange the value of an object

22650 **SYNOPSIS**

22651
```
#include <stdatomic.h>
```

22652
```
C atomic_exchange(volatile A *object, C desired);
```
22653
```
C atomic_exchange_explicit(volatile A *object, C desired,
```
22654
```
    memory_order order);
```

22655 **DESCRIPTION**

22656 CX The functionality described on this reference page is aligned with the ISO C standard. Any
22657 conflict between the requirements described here and the ISO C standard is unintentional. This
22658 volume of POSIX.1-2024 defers to the ISO C standard.

22659 Implementations that define the macro __STDC_NO_ATOMICS__ need not provide the
22660 **<stdatomic.h>** header nor support these generic functions.

22661 The *atomic_exchange_explicit*( ) generic function shall atomically replace the value pointed to by
22662 *object* with *desired*. This operation shall be an atomic read-modify-write operation (see XBD
22663 Section 4.15.1, on page 100). Memory shall be affected according to the value of *order*.

22664 The *atomic_exchange*( ) generic function shall be equivalent to *atomic_exchange_explicit*( ) called
22665 with *order* set to `memory_order_seq_cst`.

22666 **RETURN VALUE**

22667 These generic functions shall return the value pointed to by *object* immediately before the effects.

22668 **ERRORS**

22669 No errors are defined.

22670 **EXAMPLES**

22671 None.

22672 **APPLICATION USAGE**

22673 None.

22674 **RATIONALE**

22675 None.

22676 **FUTURE DIRECTIONS**

22677 None.

22678 **SEE ALSO**

22679 XBD Section 4.15.1, **<stdatomic.h>**

22680 **CHANGE HISTORY**

22681 First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

atomic_fetch_add, atomic_fetch_add_explicit, atomic_fetch_and, atomic_fetch_and_explicit, atomic_fetch_or, atomic_fetch_or_explicit, atomic_fetch_sub, atomic_fetch_sub_explicit, atomic_fetch_xor, atomic_fetch_xor_explicit — atomically replace the value of an object with the result of a computation

SYNOPSIS

```
#include <stdatomic.h>

C  atomic_fetch_add(volatile A *object, M operand);
C  atomic_fetch_add_explicit(volatile A *object, M operand,
       memory_order order);
C  atomic_fetch_and(volatile A *object, M operand);
C  atomic_fetch_and_explicit(volatile A *object, M operand,
       memory_order order);
C  atomic_fetch_or(volatile A *object, M operand);
C  atomic_fetch_or_explicit(volatile A *object, M operand,
       memory_order order);
C  atomic_fetch_sub(volatile A *object, M operand);
C  atomic_fetch_sub_explicit(volatile A *object, M operand,
       memory_order order);
C  atomic_fetch_xor(volatile A *object, M operand);
C  atomic_fetch_xor_explicit(volatile A *object, M operand,
       memory_order order);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

Implementations that define the macro __STDC_NO_ATOMICS__ need not provide the **<stdatomic.h>** header nor support these generic functions.

The *atomic_fetch_add_explicit*( ) generic function shall atomically replace the value pointed to by *object* with the result of adding *operand* to this value. This operation shall be an atomic read-modify-write operation (see XBD Section 4.15.1, on page 100). Memory shall be affected according to the value of *order*.

The *atomic_fetch_add*( ) generic function shall be equivalent to *atomic_fetch_add_explicit*( ) called with *order* set to `memory_order_seq_cst`.

The other *atomic_fetch_**( ) generic functions shall be equivalent to *atomic_fetch_add_explicit*( ) if their name ends with *explicit*, or to *atomic_fetch_add*( ) if it does not, respectively, except that they perform the computation indicated in their name, instead of addition:

sub subtraction

or bitwise inclusive OR

xor bitwise exclusive OR

and bitwise AND

For addition and subtraction, the application shall ensure that ***A*** is an atomic integer type or an atomic pointer type and is not **atomic_bool**. For the other operations, the application shall ensure that ***A*** is an atomic integer type and is not **atomic_bool**.

For signed integer types, the computation shall silently wrap around on overflow; there are no undefined results. For pointer types, the result can be an undefined address, but the

22728 computations otherwise have no undefined behavior.

22729 **RETURN VALUE**

22730 These generic functions shall return the value pointed to by *object* immediately before the effects.

22731 **ERRORS**

22732 No errors are defined.

22733 **EXAMPLES**

22734 None.

22735 **APPLICATION USAGE**

22736 The operation of these generic functions is nearly equivalent to the operation of the
22737 corresponding compound assignment operators +=, −=, etc. The only differences are that the
22738 compound assignment operators are not guaranteed to operate atomically, and the value yielded
22739 by a compound assignment operator is the updated value of the object, whereas the value
22740 returned by these generic functions is the previous value of the atomic object.

22741 **RATIONALE**

22742 None.

22743 **FUTURE DIRECTIONS**

22744 None.

22745 **SEE ALSO**

22746 XBD Section 4.15.1, **<stdatomic.h>**

22747 **CHANGE HISTORY**

22748 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

NAME

atomic_flag_clear, atomic_flag_clear_explicit — clear an atomic flag

SYNOPSIS

```
#include <stdatomic.h>

void atomic_flag_clear(volatile atomic_flag *object);
void atomic_flag_clear_explicit(volatile atomic_flag *object,
    memory_order order);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

Implementations that define the macro __STDC_NO_ATOMICS__ need not provide the **<stdatomic.h>** header nor support these generic functions.

The *atomic_flag_clear_explicit*( ) function shall atomically place the atomic flag pointed to by *object* into the clear state. Memory shall be affected according to the value of *order*, which the application shall ensure is not `memory_order_acquire` nor `memory_order_acq_rel`.

The *atomic_flag_clear*( ) function shall be equivalent to *atomic_flag_clear_explicit*( ) called with *order* set to `memory_order_seq_cst`.

RETURN VALUE

These functions shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD Section 4.15.1, **<stdatomic.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

atomic_flag_test_and_set, atomic_flag_test_and_set_explicit — test and set an atomic flag

SYNOPSIS

```
#include <stdatomic.h>

_Bool atomic_flag_test_and_set(volatile atomic_flag *object);
_Bool atomic_flag_test_and_set_explicit(volatile atomic_flag *object,
    memory_order order);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

Implementations that define the macro __STDC_NO_ATOMICS__ need not provide the **<stdatomic.h>** header nor support these generic functions.

The *atomic_flag_test_and_set_explicit*( ) function shall atomically place the atomic flag pointed to by *object* into the set state and return the value corresponding to the immediately preceding state. This operation shall be an atomic read-modify-write operation (see Section 4.15.1, on page 100). Memory shall be affected according to the value of *order*.

The *atomic_flag_test_and_set*( ) function shall be equivalent to *atomic_flag_test_and_set_explicit*( ) called with *order* set to `memory_order_seq_cst`.

RETURN VALUE

These functions shall return the value that corresponds to the state of the atomic flag immediately before the effects. The return value true shall correspond to the set state and the return value false shall correspond to the clear state.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD Section 4.15.1, **<stdatomic.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

22820 **NAME**

22821 atomic_init — initialize an atomic object

22822 **SYNOPSIS**

22823
```
#include <stdatomic.h>
```

22824
```
void atomic_init(volatile A *obj, C value);
```

22825 **DESCRIPTION**

22826 CX The functionality described on this reference page is aligned with the ISO C standard. Any
22827 conflict between the requirements described here and the ISO C standard is unintentional. This
22828 volume of POSIX.1-2024 defers to the ISO C standard.

22829 Implementations that define the macro __STDC_NO_ATOMICS__ need not provide the
22830 **<stdatomic.h>** header nor support these generic functions.

22831 The *atomic_init*( ) generic function shall initialize the atomic object pointed to by *obj* to the value
22832 *value*, while also initializing any additional state that the implementation might need to carry for
22833 the atomic object.

22834 Although this function initializes an atomic object, it does not avoid data races; concurrent
22835 access to the variable being initialized, even via an atomic operation, constitutes a data race.

22836 **RETURN VALUE**

22837 The *atomic_init*( ) generic function shall not return a value.

22838 **ERRORS**

22839 No errors are defined.

22840 **EXAMPLES**

22841
```
atomic_int guide;
```
22842
```
atomic_init(&guide, 42);
```

22843 **APPLICATION USAGE**

22844 None.

22845 **RATIONALE**

22846 None.

22847 **FUTURE DIRECTIONS**

22848 None.

22849 **SEE ALSO**

22850 XBD **<stdatomic.h>**

22851 **CHANGE HISTORY**

22852 First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

atomic_is_lock_free — indicate whether or not atomic operations are lock-free

SYNOPSIS

```
#include <stdatomic.h>

_Bool atomic_is_lock_free(const volatile A *obj);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

Implementations that define the macro __STDC_NO_ATOMICS__ need not provide the **<stdatomic.h>** header nor support these generic functions.

The *atomic_is_lock_free*( ) generic function shall indicate whether or not atomic operations on objects of the type pointed to by *obj* are lock-free; *obj* can be a null pointer.

RETURN VALUE

The *atomic_is_lock_free*( ) generic function shall return a non-zero value if and only if atomic operations on objects of the type pointed to by *obj* are lock-free. During the lifetime of the calling process, the result of the lock-free query shall be consistent for all pointers of the same type.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Operations that are lock-free should also be address-free. That is, atomic operations on the same memory location via two different addresses will communicate atomically. The implementation should not depend on any per-process state. This restriction enables communication via memory mapped into a process more than once and memory shared between two processes.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD **<stdatomic.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

atomic_load, atomic_load_explicit — atomically obtain the value of an object

SYNOPSIS

```
#include <stdatomic.h>

C atomic_load(const volatile A *object);
C atomic_load_explicit(const volatile A *object, memory_order order);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

Implementations that define the macro __STDC_NO_ATOMICS__ need not provide the **<stdatomic.h>** header nor support these generic functions.

The *atomic_load_explicit*( ) generic function shall atomically obtain the value pointed to by *object*. Memory shall be affected according to the value of *order*, which the application shall ensure is not `memory_order_release` nor `memory_order_acq_rel`.

The *atomic_load*( ) generic function shall be equivalent to *atomic_load_explicit*( ) called with *order* set to `memory_order_seq_cst`.

RETURN VALUE

These generic functions shall return the value pointed to by *object*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD Section 4.15.1, **<stdatomic.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

22920 **NAME**

22921 atomic_signal_fence, atomic_thread_fence — fence operations

22922 **SYNOPSIS**

22923 ```
#include <stdatomic.h>

void atomic_signal_fence(memory_order order);
void atomic_thread_fence(memory_order order);
```

22926 **DESCRIPTION**

22927 CX The functionality described on this reference page is aligned with the ISO C standard. Any
22928 conflict between the requirements described here and the ISO C standard is unintentional. This
22929 volume of POSIX.1-2024 defers to the ISO C standard.

22930 Implementations that define the macro __STDC_NO_ATOMICS__ need not provide the
22931 **<stdatomic.h>** header nor support these generic functions.

22932 The *atomic_signal_fence*( ) and *atomic_thread_fence*( ) functions provide synchronization primitives
22933 called *fences*. Fences can have acquire semantics, release semantics, or both. A fence with acquire
22934 semantics is called an *acquire fence*; a fence with release semantics is called a *release fence*.

22935 A release fence *A* synchronizes with an acquire fence *B* if there exist atomic operations *X* and *Y*,
22936 both operating on some atomic object *M*, such that *A* is sequenced before *X*, *X* modifies *M*, *Y* is
22937 sequenced before *B*, and *Y* reads the value written by *X* or a value written by any side effect in
22938 the hypothetical release sequence *X* would head if it were a release operation.

22939 A release fence *A* synchronizes with an atomic operation *B* that performs an acquire operation
22940 on an atomic object *M* if there exists an atomic operation *X* such that *A* is sequenced before *X*, *X*
22941 modifies *M*, and *B* reads the value written by *X* or a value written by any side effect in the
22942 hypothetical release sequence *X* would head if it were a release operation.

22943 An atomic operation *A* that is a release operation on an atomic object *M* synchronizes with an
22944 acquire fence *B* if there exists some atomic operation *X* on *M* such that *X* is sequenced before *B*
22945 and reads the value written by *A* or a value written by any side effect in the release sequence
22946 headed by *A*.

22947 Depending on the value of *order*, the operation performed by *atomic_thread_fence*( ) shall:

22948 • have no effects, if *order* is equal to `memory_order_relaxed`;

22949 • be an acquire fence, if *order* is equal to `memory_order_acquire` or
22950 `memory_order_consume`;

22951 • be a release fence, if *order* is equal to `memory_order_release`;

22952 • be both an acquire fence and a release fence, if *order* is equal to `memory_order_acq_rel`;

22953 • be a sequentially consistent acquire and release fence, if *order* is equal to
22954 `memory_order_seq_cst`.

22955 The *atomic_signal_fence*( ) function shall be equivalent to *atomic_thread_fence*( ), except that the
22956 resulting ordering constraints shall be established only between a thread and a signal handler
22957 executed in the same thread.

22958 **RETURN VALUE**

22959 These functions shall not return a value.

  
```

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *atomic_signal_fence*( ) function can be used to specify the order in which actions performed by the thread become visible to the signal handler. Implementation reorderings of loads and stores are inhibited in the same way as with *atomic_thread_fence*( ), but the hardware fence instructions that *atomic_thread_fence*( ) would have inserted are not emitted.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD Section 4.15.1, **<stdatomic.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

atomic_store, atomic_store_explicit — atomically store a value in an object

SYNOPSIS

```
#include <stdatomic.h>

void atomic_store(volatile A *object, C desired);
void atomic_store_explicit(volatile A *object, C desired,
    memory_order order);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

Implementations that define the macro __STDC_NO_ATOMICS__ need not provide the **<stdatomic.h>** header nor support these generic functions.

The *atomic_store_explicit*( ) generic function shall atomically replace the value pointed to by *object* with *desired*. Memory shall be affected according to the value of *order*, which the application shall ensure is not `memory_order_acquire`, `memory_order_consume`, nor `memory_order_acq_rel`.

The *atomic_store*( ) generic function shall be equivalent to *atomic_store_explicit*( ) called with *order* set to `memory_order_seq_cst`.

RETURN VALUE

These generic functions shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD Section 4.15.1, **<stdatomic.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

basename — return the last component of a pathname

SYNOPSIS

XSI
```
#include <libgen.h>

char *basename(char *path);
```

DESCRIPTION

The *basename*( ) function shall take the pathname pointed to by *path* and return a pointer to the final component of the pathname, deleting any trailing `'/'` characters.

If the string pointed to by *path* consists entirely of the `'/'` character, *basename*( ) shall return a pointer to the string `"/"`, except that if the string pointed to by *path* is exactly `"//"`, it is implementation-defined whether `"/"` or `"//"` is returned.

If *path* is a null pointer or points to an empty string, *basename*( ) shall return a pointer to the string `"."`.

The *basename*( ) function may modify the string pointed to by *path*, and may return a pointer into the input string. The returned pointer might be invalidated if the input string is subsequently modified or freed. If *path* is a null pointer or points to an empty string, or if the string pointed to by *path* consists entirely of the `'/'` character, the returned pointer may point to constant data that cannot be modified.

RETURN VALUE

The *basename*( ) function shall return a pointer to the final component of *path*.

The *basename*( ) function shall always be successful and no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

Using basename( )

The following program fragment returns a pointer to the value *lib*, which is the base name of **/usr/lib**.

```
#include <libgen.h>
...
char name[] = "/usr/lib";
char *base;

base = basename(name);
...
```


Sample Input and Output Strings for the basename( ) and dirname( ) Functions and the basename and dirname Utilities

basename( ) and dirname( ) Functions path Argument	String Returned by basename( )	String Returned by dirname( )	basename and dirname Utilities string Operand	Output Written by basename Utility	Output Written by dirname Utility
"usr"	"usr"	"."	usr	usr	.
"usr/"	"usr"	"."	usr/	usr	.
""	"."	"."	empty string	. or empty string	.
"/"	"/"	"/"	/	/	/
"//"	"/" or "//" (see note 1)	"/" or "//" (see note 1)	//	/ or // (see note 1)	/ or // (see note 1)
"///"	"/"	"/" or "///"	///	/	/ or ///
"/usr/"	"usr"	"/"	/usr/	usr	/
"/usr/lib"	"lib"	"/usr"	/usr/lib	lib	/usr
"//usr//lib//"	"lib"	"//usr" or "/usr" (see note 1)	//usr//lib//	lib	//usr or /usr (see note 1)
"/home//dwc//test"	"test"	"/home//dwc" or "/home/dwc"	/home//dwc//test	test	/home//dwc or /home/dwc
"/home/.././test"	"test"	"/home/../." or "/home/.."	/home/.././test	test	/home/../. or /home/..
"/home/dwc/."	"."	"/home/dwc"	/home/dwc/.	.	/home/dwc

Note

1. Whether leading **//** can be converted to **/** depends on the implementation-defined behavior of **//** (see XBD Section 4.16 (on page 105); although the *basename*( ) and *dirname*( ) functions, and *basename* and *dirname* utilities, do not themselves perform pathname resolution, their results can be passed to a function or utility which does).

APPLICATION USAGE

Note that in some circumstances (in particular, when the returned string is required to be "/" or "."), the returned pointer might point into constant data. Therefore, if the application needs to modify the returned data, it should be copied first.

RATIONALE

Earlier versions of this standard seemed to allow thread-safe and non-thread-safe implementations of *basename*( ) and *dirname*( ), but did not allow implementations to return a null pointer and require that *errno* be set when that happened. The standard now requires thread-safe behavior for both of these functions and clearly states that they are always successful.

FUTURE DIRECTIONS

None.

SEE ALSO

dirname( )

XBD **<libgen.h>**

XCU *basename*

23090 **CHANGE HISTORY**

23091 First released in Issue 4, Version 2.

23092 **Issue 5**

23093 Moved from X/OPEN UNIX extension to BASE.

23094 Normative text previously in the APPLICATION USAGE section is moved to the
23095 DESCRIPTION.

23096 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

23097 **Issue 6**

23098 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

23099 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/20 is applied, changing the
23100 DESCRIPTION to make it clear that the string referenced is the string pointed to by *path*.

23101 **Issue 7**

23102 Austin Group Interpretation 1003.1-2001 #156 is applied.

23103 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0041 [75] is applied.

23104 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0047 [656], XSH/TC2-2008/0048 [928],
23105 and XSH/TC2-2008/0049 [612] are applied.

23106 **Issue 8**

23107 Austin Group Defects 1064 and 1358 are applied, requiring *basename*( ) to be thread-safe and
23108 allowing it to return a pointer to constant data under certain conditions.

23109 Austin Group Defect 1073 is applied, changing the EXAMPLES section.

23110 Austin Group Defect 1396 is applied, changing the EXAMPLES section.

NAME

be16toh, be32toh, be64toh, htobe16, htobe32, htobe64, htole16, htole32, htole64, le16toh, le32toh, le64toh — convert values between host and specified byte order

SYNOPSIS

```
#include <endian.h>

uint16_t be16toh(uint16_t big_endian_16bits);
uint32_t be32toh(uint32_t big_endian_32bits);
uint64_t be64toh(uint64_t big_endian_64bits);

uint16_t htobe16(uint16_t host_16bits);
uint32_t htobe32(uint32_t host_32bits);
uint64_t htobe64(uint64_t host_64bits);

uint16_t htole16(uint16_t host_16bits);
uint32_t htole32(uint32_t host_32bits);
uint64_t htole64(uint64_t host_64bits);

uint16_t le16toh(uint16_t little_endian_16bits);
uint32_t le32toh(uint32_t little_endian_32bits);
uint64_t le64toh(uint64_t little_endian_64bits);
```

DESCRIPTION

These functions shall convert integer values of various sizes between host representations and representations in a specified order.

On some implementations, these functions are defined as macros.

A little-endian representation of an integer has the least significant byte stored as the first byte, with the significance of the bytes increasing as the byte address increases. A big-endian representation has the most significant byte as the first byte, with the significance of the bytes reducing as the byte address increases.

Note: Network byte order is big-endian.

For example, the **uint32_t** value 0x01020304 is represented as the four bytes 0x04, 0x03, 0x02, 0x01 on a little-endian host, and as 0x01, 0x02, 0x03, 0x04 on a big-endian host.

For each of the sizes 16, 32 and 64, the *htobeSIZE*( ) function shall convert from whatever order the host uses to big-endian representation, *htoleSIZE*( ) shall convert to little-endian representation, *beSIZEtoh*( ) shall convert from big-endian to host order, and *leSIZEtoh*( ) shall convert from little-endian to host order.

RETURN VALUE

These functions shall return an unsigned integer of the appropriate size and representation.

ERRORS

No errors are defined.

EXAMPLES

```
#include <endian.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    uint32_t val;

    if (argc > 1) {
```



```
23155               val = (uint32_t)strtoul(argv[1], NULL, 0);
23156               printf("Value: %08x\n", val);
23157               printf("As bytes:\n");
23158               union {
23159                   uint32_t asint;
23160                   unsigned char asbytes[sizeof(uint32_t)];
23161               } u;
23162               printf("Little endian: ");
23163               u.asint = htole32(val);
23164               for (int i = 0; i < sizeof(uint32_t); i++) {
23165                   printf("%02x ", u.asbytes[i]);
23166               }
23167               printf("\n");
23168               printf("Big endian   : ");
23169               u.asint = htobe32(val);
23170               for (int i = 0; i < sizeof(uint32_t); i++) {
23171                   printf("%02x ", u.asbytes[i]);
23172               }
23173               printf("\n");
23174           }
23175           return 0;
23176       }
```

APPLICATION USAGE

Since network order is defined as big-endian, the following functions are equivalent if **<arpa/inet.h>** is included:

<endian.h>	**<arpa/inet.h>**
htobe32	*htonl*
htobe16	*htons*
be32toh	*ntohl*
be16toh	*ntohs*

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

htonl( )

XBD **<arpa/inet.h>**, **<endian.h>**

CHANGE HISTORY

First released in Issue 8.

23194 **NAME**

23195 bind — bind a name to a socket

23196 **SYNOPSIS**

23197 ```
#include <sys/socket.h>
```

23198 ```
int bind(int socket, const struct sockaddr *address,
```

23199 ```
    socklen_t address_len);
```

23200 **DESCRIPTION**

23201 The *bind*( ) function shall assign a local socket address *address* to a socket identified by descriptor
23202 *socket* that has no local socket address assigned. Sockets created with the *socket*( ) function are
23203 initially unnamed; they are identified only by their address family.

23204 The *bind*( ) function takes the following arguments:

23205 *socket* Specifies the file descriptor of the socket to be bound.

23206 *address* Points to a **sockaddr** structure containing the address to be bound to the
23207 socket. The length and format of the address depend on the address family of
23208 the socket.

23209 *address_len* Specifies the length of the **sockaddr** structure pointed to by the *address*
23210 argument.

23211 The socket specified by *socket* may require the process to have appropriate privileges to use the
23212 *bind*( ) function.

23213 If the address family of the socket is AF_UNIX, the application shall ensure that a null
23214 terminator after the pathname is included in the *sun_path* member of *address* as a **sockaddr_un**
23215 structure, and that *address_len* is at least `offsetof(struct sockaddr_un, sun_path) +`
23216 `1` plus the length of the pathname. If the pathname in the *sun_path* member of *address* names an
23217 existing file, including a symbolic link, *bind*( ) shall treat the address as already in use; see
23218 ERRORS below.

23219 If the socket address cannot be assigned immediately and O_NONBLOCK is set for the file
23220 descriptor for the socket, *bind*( ) shall fail and set *errno* to [EINPROGRESS], but the assignment
23221 request shall not be aborted, and the assignment shall be completed asynchronously. Subsequent
23222 calls to *bind*( ) for the same socket, before the assignment is completed, shall fail and set *errno* to
23223 [EALREADY].

23224 When the assignment has been performed asynchronously, *pselect*( ), *select*( ), *poll*( ), and *ppoll*( )
23225 shall indicate that the file descriptor for the socket is ready for reading and writing.

23226 **RETURN VALUE**

23227 Upon successful completion, *bind*( ) shall return 0; otherwise, –1 shall be returned and *errno* set
23228 to indicate the error.

23229 **ERRORS**

23230 The *bind*( ) function shall fail if:

23231 [EADDRINUSE] The specified address is already in use.

23232 [EADDRNOTAVAIL]
23233 The specified address is not available from the local machine.

23234 [EAFNOSUPPORT]
23235 The specified address is not a valid address for the address family of the
23236 specified socket.

  
```

23237 [EALREADY] An assignment request is already in progress for the specified socket.

23238 [EBADF] The *socket* argument is not a valid file descriptor.

23239 [EINPROGRESS] O_NONBLOCK is set for the file descriptor for the socket and the assignment
23240 cannot be immediately performed; the assignment shall be performed
23241 asynchronously.

23242 [EINVAL] The socket is already bound to an address, and the protocol does not support
23243 binding to a new address; or the socket has been shut down.

23244 [ENOBUFS] Insufficient resources were available to complete the call.

23245 [ENOTSOCK] The *socket* argument does not refer to a socket.

23246 [EOPNOTSUPP] The socket type of the specified socket does not support binding to an address.

23247 If the address family of the socket is AF_UNIX, then *bind*( ) shall fail if:

23248 [EACCES] A component of the path prefix denies search permission, or the requested
23249 name requires writing in a directory with a mode that denies write
23250 permission.

23251 [EDESTADDRREQ] or [EISDIR]
23252 The *address* argument is a null pointer.

23253 [EILSEQ] The last pathname component is not a portable filename, and cannot be
23254 created in the target directory.

23255 [EIO] An I/O error occurred.

23256 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname
23257 in *address*.

23258 [ENAMETOOLONG]
23259 The length of a component of a pathname is longer than {NAME_MAX}.

23260 [ENOENT] A component of the path prefix of the pathname in *address* does not name an
23261 existing file or the pathname is an empty string.

23262 [ENOENT] or [ENOTDIR]
23263 The pathname in *address* contains at least one non-<slash> character and ends
23264 with one or more trailing <slash> characters. If the pathname without the
23265 trailing <slash> characters would name an existing file, an [ENOENT] error
23266 shall not occur.

23267 [ENOTDIR] A component of the path prefix of the pathname in *address* names an existing
23268 file that is neither a directory nor a symbolic link to a directory, or the
23269 pathname in *address* contains at least one non-<slash> character and ends with
23270 one or more trailing <slash> characters and the last pathname component
23271 names an existing file that is neither a directory nor a symbolic link to a
23272 directory.

23273 [EROFS] The name would reside on a read-only file system.

23274 The *bind*( ) function may fail if:

23275 [EACCES] The specified address is protected and the current user does not have
23276 permission to bind to it.

[EINVAL] The *address_len* argument is not a valid length for the address family.

[EISCONN] The socket is already connected.

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in *address*.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

EXAMPLES

The following code segment shows how to create a socket and bind it to a name in the AF_UNIX domain.

```
#define MY_SOCK_PATH "/somepath"

int sfd;
struct sockaddr_un my_addr;

sfd = socket(AF_UNIX, SOCK_STREAM, 0);
if (sfd == -1)
    /* Handle error */;

memset(&my_addr, '\0', sizeof(struct sockaddr_un));
                        /* Clear structure */
my_addr.sun_family = AF_UNIX;
strncpy(my_addr.sun_path, MY_SOCK_PATH, sizeof(my_addr.sun_path) -1);

if (bind(sfd, (struct sockaddr *) &my_addr,
    sizeof(struct sockaddr_un)) == -1)
    /* Handle error */;
```

APPLICATION USAGE

An application program can retrieve the assigned socket name with the *getsockname*( ) function.

For AF_UNIX sockets, some implementations support an extension where *address_len* does not have to include a null terminator for the pathname stored in *sun_path*, which in turn allows a pathname to be one byte longer. However, such usage is not portable, and carries a risk of accessing beyond the intended bounds of the pathname length.

RATIONALE

Implementations are encouraged to have *bind*( ) report an [EILSEQ] error if the last component of the address to be bound to an AF_UNIX family socket contains any bytes that have the encoded value of a <newline> character.

FUTURE DIRECTIONS

None.

SEE ALSO

connect( ), *getsockname*( ), *listen*( ), *socket*( )

XBD **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7

Austin Group Interpretation 1003.1-2001 #044 is applied, changing the ``may fail'' [ENOBUFS] error to become a ``shall fail'' error.

Austin Group Interpretation 1003.1-2001 #143 is applied.

SD5-XSH-ERN-185 is applied.

An example is added.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0042 [146], XSH/TC1-2008/0043 [146], and XSH/TC1-2008/0044 [324] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0050 [822] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defects 293 and 1482 are applied, adding the [EILSEQ] error.

Austin Group Defect 561 is applied, changing the requirements for the *sun_path* member of the **sockaddr_un** structure.

Austin Group Defect 1263 is applied, adding *ppoll*( ).

Austin Group Defect 1605 is applied, clarifying how the [EADDRINUSE] error applies to AF_UNIX sockets.

NAME

bindtextdomain, bind_textdomain_codeset, textdomain — text domain manipulation functions

SYNOPSIS

```
#include <libintl.h>

char *bindtextdomain(const char *domainname, const char *dirname);
char *bind_textdomain_codeset(const char *domainname,
    const char *codeset);
char *textdomain(const char *domainname);
```

DESCRIPTION

The *textdomain*( ) function shall set or query the name of the current text domain of the calling process. The application shall ensure that the *domainname* argument is either a null pointer (when querying), an empty string, or a string that, when used by the *gettext* family of functions to construct a pathname to a messages object, results in a valid pathname. For portable applications, it should only contain characters from the portable filename character set.

The text domain setting made by the last successful call to *textdomain*( ) shall remain in effect across subsequent calls to *setlocale*( ), *uselocale*( ), and the *gettext* family of functions.

Applications should not use text domains whose names begin with the strings `"SYS_"` or `"libc"`. These prefixes are reserved for implementation use.

The current setting of the text domain can be queried without affecting the current state of the domain by calling *textdomain*( ) with *domainname* set to a null pointer. Calling *textdomain*( ) with a *domainname* argument of an empty string shall set the text domain to the default domain, `"messages"`.

The *bindtextdomain*( ) function shall set or query the binding of a text domain to a *dirname* that is used by the *gettext* family of functions to construct a pathname to a messages object in the text domain:

- If *domainname* is a null pointer or an empty string, *bindtextdomain*( ) shall make no changes and return a null pointer without changing *errno*.
- Otherwise, if *dirname* is a non-empty string:
  - — If *domainname* is not already bound, *bindtextdomain*( ) shall bind the text domain specified by *domainname* to the pathname pointed to by *dirname* and return the bound directory pathname on success or a null pointer on failure.
  - — If *domainname* is already bound, *bindtextdomain*( ) shall replace the existing binding with the pathname pointed to by *dirname* and return the bound directory pathname on success or a null pointer on failure. On failure, the existing binding shall remain unchanged.

  It is unspecified whether the *bindtextdomain*( ) function performs pathname resolution on *dirname*, or whether that is done by the *gettext* family of functions.
- Otherwise, if *dirname* is a null pointer:
  - — If *domainname* is bound, the function shall return the bound directory pathname.
  - — If *domainname* is not bound, the function shall return the implementation-defined default directory pathname used by the *gettext* family of functions.
- Otherwise, *dirname* is an empty string and the behavior is unspecified.

If a text domain is bound to a relative pathname and the current working directory is changed after the binding is established, the pathnames used by the *gettext* family of functions to locate

messages objects for that text domain are unspecified.

The *bind_textdomain_codeset*( ) function shall set or query the binding of a text domain to the output codeset used by the *gettext* family of functions for message strings retrieved from messages objects for the text domain specified by *domainname*:

- If *domainname* is a null pointer or an empty string, *bind_textdomain_codeset*( ) shall make no changes and return a null pointer without changing *errno*.
- Otherwise, if *codeset* is a non-empty string:
  - — If *domainname* is not already bound, *bind_textdomain_codeset*( ) shall bind the text domain specified by *domainname* to the codeset pointed to by *codeset* and return the newly bound codeset on success or a null pointer on failure.
  - — If *domainname* is already bound, *bind_textdomain_codeset*( ) shall replace the existing binding with the codeset pointed to by *codeset* and return the newly bound codeset on success or a null pointer on failure. On failure, the existing binding shall remain unchanged.

  The application shall ensure that the *codeset* argument, if non-empty, is a valid codeset name that can be used as the *tocode* argument of the *iconv_open*( ) function, and that in the codeset it specifies, the <NUL> character corresponds to a single null byte.
- Otherwise, if *codeset* is a null pointer:
  - — If *domainname* is bound, the function shall return the bound codeset.
  - — If *domainname* is not bound, the function shall return the implementation-defined default codeset used by the *gettext* family of functions.
- Otherwise, *codeset* is an empty string and the behavior is unspecified.

If *codeset* is a null pointer and *domainname* is a non-empty string, *bind_textdomain_codeset*( ) shall return the current codeset for the named domain, or a null pointer if a codeset has not yet been set. The *bind_textdomain_codeset*( ) function can be called multiple times. If successfully called multiple times with the same *domainname* argument, the last such call shall override the setting made by the previous such call.

RETURN VALUE

The return value from a successful *textdomain*( ) call shall be a pointer to a string containing the current setting of the text domain. If *domainname* is a null pointer, *textdomain*( ) shall return a pointer to the string containing the current text domain. If *textdomain*( ) was not previously called and *domainname* is a null string, the name of the default text domain shall be returned. The name of the default text domain shall be the string `"messages"`. If *textdomain*( ) fails, a null pointer shall be returned and *errno* shall be set to indicate the error.

For *bindtextdomain*( ) return values see the DESCRIPTION. When *bindtextdomain*( ) is called with a non-empty *domainname* and returns a null pointer, it shall set *errno* to indicate the error. When *bindtextdomain*( ) returns a pathname for a bound text domain, the return value shall be a pointer to a copy of the *dirname* string passed to the *bindtextdomain*( ) call that created the binding. The returned string shall remain valid until the next successful call to *bindtextdomain*( ) with a non-empty *dirname* and same *domainname*. The application shall ensure that it does not modify the returned string.

A call to the *bind_textdomain_codeset*( ) function with a non-empty *domainname* argument shall return one of the following:

- The currently bound codeset name for that text domain if *codeset* is a null pointer
- The newly bound codeset if *codeset* is non-empty
- A null pointer without changing *errno* if no codeset has yet been bound for that text domain

The application shall ensure that it does not modify the returned string. A subsequent call to *bind_textdomain_codeset*( ) with a non-empty *domainname* argument might invalidate the returned pointer or overwrite the string content. The returned pointer might also be invalidated if the calling thread is terminated. If *bind_textdomain_codeset*( ) fails, a null pointer shall be returned and *errno* shall be set to indicate the error.

ERRORS

For the conditions under which *bindtextdomain*( )—if it performs pathname resolution—fails and may fail, refer to *open*( ).

In addition, the *textdomain*( ), *bindtextdomain*( ), and *bind_textdomain_codeset*( ) functions may fail if:

[ENOMEM] Insufficient memory available.

EXAMPLES

See the examples for *gettext*.

APPLICATION USAGE

A text *domainname* is limited to {TEXTDOMAIN_MAX} bytes.

Application developers are responsible for ensuring that the text domain used is not used by other applications. To minimize the chances of collision, developers can prefix text domains with their company or application name (or both) and an underscore. For example, if your application name was `"foo"` and you wanted to use the text domain `"errors"`, you could instead use the text domain `"foo_errors"`. Note that if an application can be installed with a configurable name, a text domain prefix based on the application name should change with the application name.

Specifying a relative pathname to the *bindtextdomain*( ) function should be avoided, since it may result in messages objects being searched for in a directory relative to the current working directory of the calling process; if the process calls the *chdir*( ) function, the directory searched for may also be changed.

Since pathname resolution of *dirname* might not be performed by *bindtextdomain*( ), but could be performed later by the *gettext* family of functions, and since the latter have no way to report an error, applications should verify, using for example *stat*( ), that the directory is accessible if this is desired.

RATIONALE

Although the return type of these functions ought to be **const char** *, it is **char** * to match historical practice.

Pathname resolution of the *dirname* argument passed to *bindtextdomain*( ) may be performed by *bindtextdomain*( ) itself or by the *gettext* family of functions. If pathname resolution fails in one of the *gettext* family of functions, it is neither allowed to modify *errno* nor to return an error, but if pathname resolution fails in *bindtextdomain*( ), it is required to report an error and set *errno* just like *open*( ) does.

Historically, *bindtextdomain*( ) did not perform pathname resolution. However, the standard developers decided to allow this as an option so that future implementations can, if desired, open a file descriptor for that directory in *bindtextdomain*( ) and then use that file descriptor with

openat( ) in the *gettext* family of functions.

The *dirname* parameter to *bindtextdomain*( ) may need to be copied to avoid the possibility of the application releasing the memory used by the argument while the *gettext* family of functions may still need to reference it.

When *bindtextdomain*( ) is called with a non-empty *domainname* and an empty *dirname*, historical implementations of the *gettext* family of functions use the empty string for the *dirname* part of the messages object pathname, resulting in an absolute pathname of the form */localename/categoryname/textdomainname***.mo**. The standard developers did not believe this behavior to be useful. Using the empty *dirname* case as a way to remove an existing binding seemed to be a more useful behavior, and would be consistent with the behavior of *textdomain*( ). However, because no historical implementations behave this way, the behavior is left unspecified.

Some implementations set *errno* to [EAGAIN] to signal memory allocation failures that might succeed if retried and [ENOMEM] for failures that are unlikely to ever succeed, for example due to configured limits. Section 2.3 (on page 507) permits this behavior; when multiple error conditions are simultaneously true there is no precedence between them.

FUTURE DIRECTIONS

A future version of this standard may require implementations to prefix implementation-provided text domains with either `"SYS_"` or a prefix related to the implementor's company name to avoid namespace collisions.

A future version of this standard may require *bindtextdomain*( ) to remove any binding for *domainname* when called with a non-empty *domainname* and an empty *dirname*.

SEE ALSO

gettext, *iconv_open*( ), *setlocale*( ), *uselocale*( )

XBD **<libintl.h>**, **<limits.h>**

XCU *msgfmt*, *xgettext*

CHANGE HISTORY

First released in Issue 8.

NAME

bsearch — binary search a sorted table

SYNOPSIS

```
#include <stdlib.h>

void *bsearch(const void *key, const void *base, size_t nel,
    size_t width, int (*compar)(const void *, const void *));
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *bsearch*( ) function shall search an array of *nel* objects, the initial element of which is pointed to by *base*, for an element that matches the object pointed to by *key*. The size of each element in the array is specified by *width*. If the *nel* argument has the value zero, the comparison function pointed to by *compar* shall not be called and no match shall be found.

The comparison function pointed to by *compar* shall be called with two arguments that point to the *key* object and to an array element, in that order.

The application shall ensure that the comparison function pointed to by *compar* does not alter the contents of the array. The implementation may reorder elements of the array between calls to the comparison function, but shall not alter the contents of any individual element.

The implementation shall ensure that the first argument is always a pointer to the key.

When the same objects (consisting of width bytes, irrespective of their current positions in the array) are passed more than once to the comparison function, the results shall be consistent with one another. That is, the same object shall always compare the same way with the key.

The application shall ensure that the function returns an integer less than, equal to, or greater than 0 if the *key* object is considered, respectively, to be less than, to match, or to be greater than the array element. The application shall ensure that the array consists of all the elements that compare less than, all the elements that compare equal to, and all the elements that compare greater than the *key* object, in that order.

RETURN VALUE

The *bsearch*( ) function shall return a pointer to a matching member of the array, or a null pointer if no match is found. If two or more members compare equal, which member is returned is unspecified.

ERRORS

No errors are defined.

EXAMPLES

The example below searches a table containing pointers to nodes consisting of a string and its length. The table is ordered alphabetically on the string in the node pointed to by each entry.

The code fragment below reads in strings and either finds the corresponding node and prints out the string and its length, or prints an error message.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define TABSIZE    1000

struct node {                   /* These are stored in the table. */
```



```
        char *string;
        int length;
    };
    struct node table[TABSIZE];    /* Table to be searched. */
        .
        .
        .
    {
        struct node *node_ptr, node;
        /* Routine to compare 2 nodes. */
        int node_compare(const void *, const void *);
        .
        .
        .
        while (scanf("%ms", &node.string) != EOF) {
            node_ptr = (struct node *)bsearch((void *)(&node),
                   (void *)table, TABSIZE,
                   sizeof(struct node), node_compare);
            if (node_ptr != NULL) {
                (void)printf("string = %20s, length = %d\n",
                    node_ptr->string, node_ptr->length);
            } else {
                (void)printf("not found: %s\n", node.string);
            }
            free(node.string);
        }
    }
    /*
        This routine compares two nodes based on an
        alphabetical ordering of the string field.
    */
    int
    node_compare(const void *node1, const void *node2)
    {
        return strcoll(((const struct node *)node1)->string,
            ((const struct node *)node2)->string);
    }
```

APPLICATION USAGE

The pointers to the key and the element at the base of the table should be of type pointer-to-element.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

In practice, the array is usually sorted according to the comparison function.

RATIONALE

The requirement that the second argument (hereafter referred to as *p*) to the comparison function is a pointer to an element of the array implies that for every call all of the following expressions are non-zero:

```
( (char *)p - (char *)base ) % width == 0
(char *)p >= (char *)base
```



```
        (char *)p < (char *)base + nel * width
```

FUTURE DIRECTIONS

None.

SEE ALSO

hcreate( ), *lsearch*( ), *qsort*( ), *tdelete*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/11 is applied, adding to the DESCRIPTION the last sentence of the first non-shaded paragraph, and the following three paragraphs. The RATIONALE section is also updated. These changes are for alignment with the ISO C standard.

Issue 7

The EXAMPLES section is revised.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0051 [756] is applied.

NAME

btowc — single byte to wide character conversion

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

wint_t btowc(int c);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *btowc*( ) function shall determine whether *c* constitutes a valid (one-byte) character in the initial shift state.

The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

RETURN VALUE

The *btowc*( ) function shall return WEOF if *c* has the value EOF or if **(unsigned char)** *c* does not constitute a valid (one-byte) character in the initial shift state. Otherwise, it shall return the wide-character representation of that character.

CX In the POSIX locale, *btowc*( ) shall not return WEOF if *c* has a value in the range 0 to 255 inclusive.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wctob( )

XBD **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0052 [663] is applied.

NAME

c16rtomb, c32rtomb — convert a Unicode character code to a character (restartable)

SYNOPSIS

```
#include <uchar.h>

size_t c16rtomb(char *restrict s, char16_t c16, mbstate_t *restrict ps);
size_t c32rtomb(char *restrict s, char32_t c32, mbstate_t *restrict ps);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

If *s* is a null pointer, the *c16rtomb*( ) function shall be equivalent to the call:

```
c16rtomb(buf, L'\0', ps)
```

where *buf* is an internal buffer.

If *s* is not a null pointer, the *c16rtomb*( ) function shall determine the number of bytes needed to represent the character that corresponds to the wide character given by *c16* (including any shift sequences), and store the resulting bytes in the array whose first element is pointed to by *s*. At most {MB_CUR_MAX} bytes shall be stored. If *c16* is a null wide character, a null byte shall be stored, preceded by any shift sequence needed to restore the initial shift state; the resulting state described shall be the initial conversion state.

If *ps* is a null pointer, the *c16rtomb*( ) function shall use its own internal **mbstate_t** object, which shall be initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence.

The behavior of this function is affected by the *LC_CTYPE* category of the current locale.

The *mbrtoc16*( ) function shall not change the setting of *errno* if successful.

The *c32rtomb*( ) function shall behave the same way as *c16rtomb*( ) except that the second parameter shall be an object of type **char32_t** instead of **char16_t**. References to *c16* in the above description shall apply as if they were *c32* when they are being read as describing *c32rtomb*( ).

If called with a null *ps* argument, the *c16rtomb*( ) function need not be thread-safe; however, such calls shall avoid data races with calls to *c16rtomb*( ) with a non-null argument and with calls to all other functions.

If called with a null *ps* argument, the *c32rtomb*( ) function need not be thread-safe; however, such calls shall avoid data races with calls to *c32rtomb*( ) with a non-null argument and with calls to all other functions.

The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls *c16rtomb*( ) or *c32rtomb*( ) with a null pointer for *ps*.

RETURN VALUE

These functions shall return the number of bytes stored in the array object (including any shift sequences). When *c16* or *c32* is not a valid wide character, an encoding error shall occur. In this case, the function shall store the value of the macro [EILSEQ] in *errno* and shall return (**size_t**)−1; the conversion state is unspecified.

23684 **ERRORS**

23685 These functions shall fail if:

23686 [EILSEQ] An invalid wide-character code is detected.

23687 These functions may fail if:

23688 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

23689 **EXAMPLES**

23690 None.

23691 **APPLICATION USAGE**

23692 None.

23693 **RATIONALE**

23694 None.

23695 **FUTURE DIRECTIONS**

23696 None.

23697 **SEE ALSO**

23698 *mbrtoc16( )*

23699 XBD **<uchar.h>**

23700 **CHANGE HISTORY**

23701 First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

cabs, cabsf, cabsl — return a complex absolute value

SYNOPSIS

```
#include <complex.h>

double cabs(double complex z);
float cabsf(float complex z);
long double cabsl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the complex absolute value (also called norm, modulus, or magnitude) of *z*.

RETURN VALUE

These functions shall return the complex absolute value.

MXC *cabs*($x + iy$), *cabs*($y + ix$), and *cabs*($x - iy$) shall return exactly the same value.

If *z* is $\pm 0 \pm i0$, +0 shall be returned.

If the real or imaginary part of *z* is ±Inf, +Inf shall be returned, even if the other part is NaN.

If the real or imaginary part of *z* is NaN and the other part is not ±Inf, NaN shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

NAME

cacos, cacosf, cacosl — complex arc cosine functions

SYNOPSIS

```
#include <complex.h>

double complex cacos(double complex z);
float complex cacosf(float complex z);
long double complex cacosl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the complex arc cosine of z, with branch cuts outside the interval [–1, +1] along the real axis.

RETURN VALUE

These functions shall return the complex arc cosine value, in the range of a strip mathematically unbounded along the imaginary axis and in the interval [0, π] along the real axis.

MXC *cacos*(*conj*(*z*)), *cacosf*(*conjf*(*z*)), and *cacosl*(*conjl*(*z*)) shall return exactly the same value as *conj*(*cacos*(*z*)), *conjf*(*cacosf*(*z*)), and *conjl*(*cacosl*(*z*)), respectively, including for the special values of z below.

If z is $\pm 0 + i0$, $\pi/2 - i0$ shall be returned.

If z is $\pm 0 + i$NaN, $\pi/2 + i$NaN shall be returned.

If z is $x + i$Inf where x is finite, $\pi/2 - i$Inf shall be returned.

If z is $x + i$NaN where x is non-zero and finite, NaN + iNaN shall be returned and the invalid floating-point exception may be raised.

If z is –Inf + iy where y is positive-signed and finite, $\pi - i$Inf shall be returned.

If z is +Inf + iy where y is positive-signed and finite, +0 – iInf shall be returned.

If z is –Inf + iInf, $3\pi/4 - i$Inf shall be returned.

If z is +Inf + iInf, $\pi/4 - i$Inf shall be returned.

If z is ±Inf + iNaN, NaN ± iInf shall be returned; the sign of the imaginary part of the result is unspecified.

If z is NaN + iy where y is finite, NaN + iNaN shall be returned and the invalid floating-point exception may be raised.

If z is NaN + iInf, NaN – iInf shall be returned.

If z is NaN + iNaN, NaN – iNaN shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ccos( )

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

NAME

cacosh, cacoshf, cacoshl — complex arc hyperbolic cosine functions

SYNOPSIS

```
#include <complex.h>

double complex cacosh(double complex z);
float complex cacoshf(float complex z);
long double complex cacoshl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the complex arc hyperbolic cosine of z, with a branch cut at values less than 1 along the real axis.

RETURN VALUE

These functions shall return the complex arc hyperbolic cosine value, in the range of a half-strip of non-negative values along the real axis and in the interval $[-i\pi, +i\pi]$ along the imaginary axis.

MXC *cacosh*(*conj*(*z*)), *cacoshf*(*conjf*(*z*)), and *cacoshl*(*conjl*(*z*)) shall return exactly the same value as *conj*(*cacosh*(*z*)), *conjf*(*cacoshf*(*z*)), and *conjl*(*cacoshl*(*z*)), respectively, including for the special values of z below.

If z is $\pm 0 + i0$, $+0 + i\pi/2$ shall be returned.

If z is $x + i\text{Inf}$ where x is finite, $+\text{Inf} + i\pi/2$ shall be returned.

If z is $0 + i\text{NaN}$, $\text{NaN} \pm i\pi/2$ shall be returned; the sign of the imaginary part of the result is unspecified.

If z is $x + i\text{NaN}$ where x is non-zero and finite, $\text{NaN} + i\text{NaN}$ shall be returned and the invalid floating-point exception may be raised.

If z is $-\text{Inf} + iy$ where y is positive-signed and finite, $+\text{Inf} + i\pi$ shall be returned.

If z is $+\text{Inf} + iy$ where y is positive-signed and finite, $+\text{Inf} + i0$ shall be returned.

If z is $-\text{Inf} + i\text{Inf}$, $+\text{Inf} + i3\pi/4$ shall be returned.

If z is $+\text{Inf} + i\text{Inf}$, $+\text{Inf} + i\pi/4$ shall be returned.

If z is $\pm\text{Inf} + i\text{NaN}$, $+\text{Inf} + i\text{NaN}$ shall be returned.

If z is $\text{NaN} + iy$ where y is finite, $\text{NaN} + i\text{NaN}$ shall be returned and the invalid floating-point exception may be raised.

If z is $\text{NaN} + i\text{Inf}$, $+\text{Inf} + i\text{NaN}$ shall be returned.

If z is $\text{NaN} + i\text{NaN}$, $\text{NaN} + i\text{NaN}$ shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ccosh( )

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

23842 **NAME**

23843 cacosl — complex arc cosine functions

23844 **SYNOPSIS**

23845 `#include <complex.h>`

23846 `long double complex cacosl(long double complex z);`

23847 **DESCRIPTION**

23848 Refer to *cacos( )*.

NAME

call_once — dynamic package initialization

SYNOPSIS

```
#include <threads.h>

void call_once(once_flag *flag, void (*init_routine)(void));
once_flag flag = ONCE_FLAG_INIT;
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *call_once*( ) function shall use the **once_flag** pointed to by *flag* to ensure that *init_routine* is called exactly once, the first time the *call_once*( ) function is called with that value of *flag*. Completion of an effective call to the *call_once*( ) function shall synchronize with all subsequent calls to the *call_once*( ) function with the same value of *flag*.

CX The *call_once*( ) function is not a cancellation point. However, if *init_routine* is a cancellation point and is canceled, the effect on *flag* shall be as if *call_once*( ) was never called.

If the call to *init_routine* is terminated by a call to *longjmp*( ) or *siglongjmp*( ), the behavior is undefined.

The behavior of *call_once*( ) is undefined if *flag* has automatic storage duration or is not initialized by ONCE_FLAG_INIT.

The *call_once*( ) function shall not be affected if the calling thread executes a signal handler during the call.

RETURN VALUE

The *call_once*( ) function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

If *init_routine* recursively calls *call_once*( ) with the same *flag*, the recursive call will not call the specified *init_routine*, and thus the specified *init_routine* will not complete, and thus the recursive call to *call_once*( ) will not return. Use of *longjmp*( ) or *siglongjmp*( ) within an *init_routine* to jump to a point outside of *init_routine* prevents *init_routine* from returning.

RATIONALE

For dynamic library initialization in a multi-threaded process, if an initialization flag is used the flag needs to be protected against modification by multiple threads simultaneously calling into the library. This can be done by using a statically-initialized mutex. However, the better solution is to use *call_once*( ) or *pthread_once*( ) which are designed for exactly this purpose, for example:

```
#include <threads.h>
static once_flag random_is_initialized = ONCE_FLAG_INIT;
extern void initialize_random(void);

int random_function()
{
    call_once(&random_is_initialized, initialize_random);
    ...
```



```
        /* Operations performed after initialization. */
    }
```

The *call_once*( ) function is not affected by signal handlers for the reasons stated in XRAT Section B.2.3 (on page 3742).

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_once( )

XBD Section 4.15.2, **<threads.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

23905 **NAME**

23906 calloc — a memory allocator

23907 **SYNOPSIS**

23908 ```
#include <stdlib.h>
```

23909 ```
void *calloc(size_t nelem, size_t elsize);
```

23910 **DESCRIPTION**

23911 CX The functionality described on this reference page is aligned with the ISO C standard. Any
23912 conflict between the requirements described here and the ISO C standard is unintentional. This
23913 volume of POSIX.1-2024 defers to the ISO C standard.

23914 The *calloc*( ) function shall allocate unused space for an array of *nelem* elements each of whose
23915 size in bytes is *elsize*. The space shall be initialized to all bits 0.

23916 The order and contiguity of storage allocated by successive calls to *calloc*( ) is unspecified. The
23917 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to
23918 a pointer to any type of object with a fundamental alignment requirement and then used to
23919 access such an object or an array of such objects in the space allocated (until the space is
23920 explicitly freed or reallocated). Each such allocation shall yield a pointer to an object disjoint
23921 from any other object. The pointer returned shall point to the start (lowest byte address) of the
23922 allocated space. If the space cannot be allocated, a null pointer shall be returned. If the size of the
23923 space requested is 0, the behavior is implementation-defined: either a null pointer shall be
23924 returned, or the behavior shall be as if the size were some non-zero value, except that the
23925 behavior is undefined if the returned pointer is used to access an object.

23926 For purposes of determining the existence of a data race, *calloc*( ) shall behave as though it
23927 accessed only memory locations accessible through its arguments and not other static duration
23928 storage. The function may, however, visibly modify the storage that it allocates. Calls to
23929 ADV *aligned_alloc*( ), *calloc*( ), *free*( ), *malloc*( ), *posix_memalign*( ),
23930 CX *reallocarray*( ), and *realloc*( ) that allocate or deallocate a particular region of memory shall occur
23931 in a single total order (see Section 4.15.1, on page 100), and each such deallocation call shall
23932 synchronize with the next allocation (if any) in this order.

23933 **RETURN VALUE**

23934 Upon successful completion, *calloc*( ) shall return a pointer to the allocated space; if either *nelem*
23935 or *elsize* is 0, the application shall ensure that the pointer is not used to access an object.

23936 CX Otherwise, it shall return a null pointer and set *errno* to indicate the error.

23937 **ERRORS**

23938 The *calloc*( ) function shall fail if:

23939 CX [ENOMEM] Insufficient memory is available, including the case when *nelem* * *elsize* would
23940 overflow.

23941 The *calloc*( ) function may fail if:

23942 CX [EINVAL] *nelem* or *elsize* is 0 and the implementation does not support 0 sized
23943 allocations.

EXAMPLES

None.

APPLICATION USAGE

There is now no requirement for the implementation to support the inclusion of **<malloc.h>**.

RATIONALE

See the RATIONALE for *malloc( )*.

FUTURE DIRECTIONS

None.

SEE ALSO

aligned_alloc( ), *free( )*, *malloc( )*, *realloc( )*

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The setting of *errno* and the [ENOMEM] error condition are mandatory if an insufficient memory condition occurs.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0053 [526] is applied.

Issue 8

Austin Group Defect 374 is applied, changing the RETURN VALUE and ERRORS sections in relation to 0 sized allocations.

Austin Group Defect 1218 is applied, changing the [ENOMEM] error.

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018 standard.

Austin Group Defect 1387 is applied, changing the RATIONALE section.

NAME

carg, cargf, cargl — complex argument functions

SYNOPSIS

```
#include <complex.h>

double carg(double complex z);
float cargf(float complex z);
long double cargl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the argument (also called phase angle) of z, with a branch cut along the negative real axis.

RETURN VALUE

These functions shall return the value of the argument in the interval $[-\pi, +\pi]$.

MXC If z is $-0 \pm i0$, $\pm\pi$ shall be returned.

If z is $+0 \pm i0$, ± 0 shall be returned.

If z is $x \pm i0$ where x is negative, $\pm\pi$ shall be returned.

If z is $x \pm i0$ where x is positive, ± 0 shall be returned.

If z is $\pm 0 + iy$ where y is negative, $-\pi/2$ shall be returned.

If z is $\pm 0 + iy$ where y is positive, $\pi/2$ shall be returned.

If z is $-\text{Inf} \pm iy$ where y is positive and finite, $\pm\pi$ shall be returned.

If z is $+\text{Inf} \pm iy$ where y is positive and finite, ± 0 shall be returned.

If z is $x \pm i\text{Inf}$ where x is finite, $\pm\pi/2$ shall be returned.

If z is $-\text{Inf} \pm i\text{Inf}$, $\pm 3\pi/4$ shall be returned.

If z is $+\text{Inf} \pm i\text{Inf}$, $\pm\pi/4$ shall be returned.

If the real or imaginary part of z is NaN, NaN shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cimag( ), *conj( )*, *cproj( )*

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

NAME

casin, casinf, casinl — complex arc sine functions

SYNOPSIS

```
#include <complex.h>

double complex casin(double complex z);
float complex casinf(float complex z);
long double complex casinl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the complex arc sine of *z*, with branch cuts outside the interval [–1, +1] along the real axis.

RETURN VALUE

These functions shall return the complex arc sine value, in the range of a strip mathematically unbounded along the imaginary axis and in the interval $[-\pi/2, +\pi/2]$ along the real axis.

MXC *casin*(*conj*(*iz*)), *casinf*(*conjf*(*iz*)), and *casinl*(*conjl*(*iz*)) shall return exactly the same value as *conj*(*casin*(*iz*)), *conjf*(*casinf*(*iz*)), and *conjl*(*casinl*(*iz*)), respectively, and *casin*(–*iz*), *casinf*(–*iz*), and *casinl*(–*iz*) shall return exactly the same value as –*casin*(*iz*), –*casinf*(*iz*), and –*casinl*(*iz*), respectively, including for the special values of *iz* below.

If *iz* is +0 + *i*0, –*i* (0 + *i*0) shall be returned.

If *iz* is *x* + *i*Inf where *x* is positive-signed and finite, –*i* (+Inf + $i\pi/2$) shall be returned.

If *iz* is *x* + iNaN where *x* is finite, –*i* (NaN + *i*NaN) shall be returned and the invalid floating-point exception may be raised.

If *iz* is +Inf + *iy* where *y* is positive-signed and finite, –*i* (+Inf + *i*0) shall be returned.

If *iz* is +Inf + *i*Inf, –*i* (+Inf + $i\pi/4$) shall be returned.

If *iz* is +Inf + *i*NaN, –*i* (+Inf + *i*NaN) shall be returned.

If *iz* is NaN + *i*0, –*i* (NaN + *i*0) shall be returned.

If *iz* is NaN + *iy* where *y* is non-zero and finite, –*i* (NaN + *i*NaN) shall be returned and the invalid floating-point exception may be raised.

If *iz* is NaN + *i*Inf, –*i* (±Inf + *i*NaN) shall be returned; the sign of the imaginary part of the result is unspecified.

If *iz* is NaN + *i*NaN, –*i* (NaN + *i*NaN) shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The MXC special cases for *casin*( ) are derived from those for *casinh*( ) by applying the formula *casin*(*z*) = –*i casinh*(*iz*).

FUTURE DIRECTIONS

None.

SEE ALSO

casinh( ), *csin*( )

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

NAME

casinh, casinhf, casinhl — complex arc hyperbolic sine functions

SYNOPSIS

```
#include <complex.h>

double complex casinh(double complex z);
float complex casinhf(float complex z);
long double complex casinhl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the complex arc hyperbolic sine of z, with branch cuts outside the interval $[-i, +i]$ along the imaginary axis.

RETURN VALUE

These functions shall return the complex arc hyperbolic sine value, in the range of a strip mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the imaginary axis.

MXC *casinh*(*conj*(*z*)), *casinhf* (*conjf* (*z*)), and *casinhl*(*conjl*(*z*)) shall return exactly the same value as *conj*(*casinh*(*z*)), *conjf* (*casinhf* (*z*)), and *conjl*(*casinhl*(*z*)), respectively, and *casinh*(–*z*), *casinhf* (–*z*), and *casinhl*(–*z*) shall return exactly the same value as –*casinh*(*z*), –*casinhf* (*z*), and –*casinhl*(*z*), respectively, including for the special values of z below.

If z is $+0 + i0$, $0 + i0$ shall be returned.

If z is $x + i\text{Inf}$ where x is positive-signed and finite, $+\text{Inf} + i\pi/2$ shall be returned.

If z is $x + i\text{NaN}$ where x is finite, $\text{NaN} + i\text{NaN}$ shall be returned and the invalid floating-point exception may be raised.

If z is $+\text{Inf} + iy$ where y is positive-signed and finite, $+\text{Inf} + i0$ shall be returned.

If z is $+\text{Inf} + i\text{Inf}$, $+\text{Inf} + i\pi/4$ shall be returned.

If z is $+\text{Inf} + i\text{NaN}$, $+\text{Inf} + i\text{NaN}$ shall be returned.

If z is $\text{NaN} + i0$, $\text{NaN} + i0$ shall be returned.

If z is $\text{NaN} + iy$ where y is non-zero and finite, $\text{NaN} + i\text{NaN}$ shall be returned and the invalid floating-point exception may be raised.

If z is $\text{NaN} + i\text{Inf}$, $\pm\text{Inf} + i\text{NaN}$ shall be returned; the sign of the real part of the result is unspecified.

If z is $\text{NaN} + i\text{NaN}$, $\text{NaN} + i\text{NaN}$ shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

csinh( )

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

24121 **NAME**

24122 casinl — complex arc sine functions

24123 **SYNOPSIS**

24124 `#include <complex.h>`

24125 `long double complex casinl(long double complex z);`

24126 **DESCRIPTION**

24127 Refer to *casin( )*.

NAME

catan, catanf, catanl — complex arc tangent functions

SYNOPSIS

```
#include <complex.h>

double complex catan(double complex z);
float complex catanf(float complex z);
long double complex catanl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the complex arc tangent of z, with branch cuts outside the interval $[-i, +i]$ along the imaginary axis.

RETURN VALUE

These functions shall return the complex arc tangent value, in the range of a strip mathematically unbounded along the imaginary axis and in the interval $[-\pi/2, +\pi/2]$ along the real axis.

MXC *catan*(*conj*(*iz*)), *catanf*(*conjf*(*iz*)), and *catanl*(*conjl*(*iz*)) shall return exactly the same value as *conj*(*catan*(*iz*)), *conjf*(*catanf*(*iz*)), and *conjl*(*catanl*(*iz*)), respectively, and *catan*(–*iz*), *catanf*(–*iz*), and *catanl*(–*iz*) shall return exactly the same value as –*catan*(*iz*), –*catanf*(*iz*), and –*catanl*(*iz*), respectively, including for the special values of *iz* below.

If *iz* is +0 + *i*0, –*i* (+0 + *i*0) shall be returned.

If *iz* is +0 + *i*NaN, –*i* (+0 + *i*NaN) shall be returned.

If *iz* is +1 + *i*0, –*i* (+Inf + *i*0) shall be returned and the divide-by-zero floating-point exception shall be raised.

If *iz* is *x* + *i*Inf where *x* is positive-signed and finite, –*i* (+0 + *i*$\pi/2$) shall be returned.

If *iz* is *x* + *i*NaN where *x* is non-zero and finite, –*i* (NaN + *i*NaN) shall be returned and the invalid floating-point exception may be raised.

If *iz* is +Inf + *iy* where *y* is positive-signed and finite, –*i* (+0 + *i*$\pi/2$) shall be returned.

If *iz* is +Inf + *i*Inf, –*i* (+0 + *i*$\pi/2$) shall be returned.

If *iz* is +Inf + *i*NaN, –*i* (+0 + *i*NaN) shall be returned.

If *iz* is NaN + *iy* where *y* is finite, –*i* (NaN + *i*NaN) shall be returned and the invalid floating-point exception may be raised.

If *iz* is NaN + *i*Inf, –*i* (±0 + *i*$\pi/2$) shall be returned; the sign of the imaginary part of the result is unspecified.

If *iz* is NaN + *i*NaN, –*i* (NaN + *i*NaN) shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The MXC special cases for *catan*( ) are derived from those for *catanh*( ) by applying the formula *catan*(*z*) = −*i* *catanh*(*i**z*).

FUTURE DIRECTIONS

None.

SEE ALSO

catanh( ), *ctan*( )

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

NAME

catanh, catanhf, catanhl — complex arc hyperbolic tangent functions

SYNOPSIS

```
#include <complex.h>

double complex catanh(double complex z);
float complex catanhf(float complex z);
long double complex catanhl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the complex arc hyperbolic tangent of z, with branch cuts outside the interval [–1, +1] along the real axis.

RETURN VALUE

These functions shall return the complex arc hyperbolic tangent value, in the range of a strip mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the imaginary axis.

MXC *catanh*(*conj*(*z*)), *catanhf*(*conjf*(*z*)), and *catanhl*(*conjl*(*z*)) shall return exactly the same value as *conj*(*catanh*(*z*)), *conjf*(*catanhf*(*z*)), and *conjl*(*catanhl*(*z*)), respectively, and *catanh*(–*z*), *catanhf*(–*z*), and *catanhl*(–*z*) shall return exactly the same value as –*catanh*(*z*), –*catanhf*(*z*), and –*catanhl*(*z*), respectively, including for the special values of z below.

If z is +0 + i0, +0 + i0 shall be returned.

If z is +0 + iNaN, +0 + iNaN shall be returned.

If z is +1 + i0, +Inf + i0 shall be returned and the divide-by-zero floating-point exception shall be raised.

If z is x + iInf where x is positive-signed and finite, +0 + $i\pi/2$ shall be returned.

If z is x + iNaN where x is non-zero and finite, NaN + iNaN shall be returned and the invalid floating-point exception may be raised.

If z is +Inf + iy where y is positive-signed and finite, +0 + $i\pi/2$ shall be returned.

If z is +Inf + iInf, +0 + $i\pi/2$ shall be returned.

If z is +Inf + iNaN, +0 + iNaN shall be returned.

If z is NaN + iy where y is finite, NaN + iNaN shall be returned and the invalid floating-point exception may be raised.

If z is NaN + iInf, ±0 + $i\pi/2$ shall be returned; the sign of the real part of the result is unspecified.

If z is NaN + iNaN, NaN + iNaN shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ctanh( )

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

24236 **NAME**

24237 catanl — complex arc tangent functions

24238 **SYNOPSIS**

24239 `#include <complex.h>`

24240 `long double complex catanl(long double complex z);`

24241 **DESCRIPTION**

24242 Refer to *catan( )*.

NAME

catclose — close a message catalog descriptor

SYNOPSIS

```
#include <nl_types.h>

int catclose(nl_catd catd);
```

DESCRIPTION

The *catclose*( ) function shall close the message catalog identified by *catd*. If a file descriptor is used to implement the type **nl_catd**, that file descriptor shall be closed.

RETURN VALUE

Upon successful completion, *catclose*( ) shall return 0; otherwise, –1 shall be returned, and *errno* set to indicate the error.

ERRORS

The *catclose*( ) function may fail if:

[EBADF] The catalog descriptor is not valid.

[EINTR] The *catclose*( ) function was interrupted by a signal.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

catgets( ), *catopen*( )

XBD **<nl_types.h>**

CHANGE HISTORY

First released in Issue 2.

Issue 7

The *catclose*( ) function is moved from the XSI option to the Base.

NAME

catgets — read a program message

SYNOPSIS

```
#include <nl_types.h>

char *catgets(nl_catd catd, int set_id, int msg_id, const char *s);
```

DESCRIPTION

The *catgets*( ) function shall attempt to read message *msg_id*, in set *set_id*, from the message catalog identified by *catd*. The *catd* argument is a message catalog descriptor returned from an earlier call to *catopen*( ). The results are undefined if *catd* is not a value returned by *catopen*( ) for a message catalog still open in the process. The *s* argument points to a default message string which shall be returned by *catgets*( ) if it cannot retrieve the identified message.

The *catgets*( ) function need not be thread-safe.

RETURN VALUE

If the identified message is retrieved successfully, *catgets*( ) shall return a pointer to an internal buffer area containing the null-terminated message string. If the call is unsuccessful for any reason, *s* shall be returned and *errno* shall be set to indicate the error.

ERRORS

The *catgets*( ) function shall fail if:

[EINTR] The read operation was terminated due to the receipt of a signal, and no data was transferred.

[ENOMSG] The message identified by *set_id* and *msg_id* is not in the message catalog.

The *catgets*( ) function may fail if:

[EBADF] The *catd* argument is not a valid message catalog descriptor open for reading.

[EBADMSG] The message identified by *set_id* and *msg_id* in the specified message catalog did not satisfy implementation-defined security criteria.

[EINVAL] The message catalog identified by *catd* is corrupted.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

catclose( ), *catopen*( )

XBD **<nl_types.h>**

CHANGE HISTORY

First released in Issue 2.

Issue 5

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

Issue 6

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

Issue 7

Austin Group Interpretation 1003.1-2001 #044 is applied, changing the ``may fail'' [EINTR] and [ENOMSG] errors to become ``shall fail'' errors, updating the RETURN VALUE section, and updating the DESCRIPTION to note that: ``The results are undefined if *catd* is not a value returned by *catopen*( ) for a message catalog still open in the process.''

The *catgets*( ) function is moved from the XSI option to the Base.

NAME

catopen — open a message catalog

SYNOPSIS

```
#include <nl_types.h>

nl_catd catopen(const char *name, int oflag);
```

DESCRIPTION

The *catopen*( ) function shall open a message catalog and return a message catalog descriptor. The *name* argument specifies the name of the message catalog to be opened. If *name* contains a '/', then *name* specifies a pathname for the message catalog. Otherwise, the environment variable *NLSPATH* is used with *name* substituted for the %N conversion specification (see XBD Chapter 8, on page 167); if *NLSPATH* exists in the environment when the process starts, then if the process has appropriate privileges, the behavior of *catopen*( ) is undefined. If *NLSPATH* does not exist in the environment, or if a message catalog cannot be found in any of the components specified by *NLSPATH*, then an implementation-defined default path shall be used. This default may be affected by the setting of *LC_MESSAGES* if the value of *oflag* is NL_CAT_LOCALE, or the *LANG* environment variable if *oflag* is 0. When searching *NLSPATH*, *catopen*( ) shall ignore any files it finds that are not valid message catalog files.

A message catalog descriptor shall remain valid in a process until that process closes it, or a successful call to one of the *exec* functions. A change in the setting of the *LC_MESSAGES* category may invalidate existing open catalogs.

If a file descriptor is used to implement message catalog descriptors, the FD_CLOEXEC flag shall be set; see **<fcntl.h>**.

If the value of the *oflag* argument is 0, the *LANG* environment variable is used to locate the catalog without regard to the *LC_MESSAGES* category. If the *oflag* argument is NL_CAT_LOCALE, the *LC_MESSAGES* category is used to locate the message catalog (see XBD Section 8.2, on page 169).

RETURN VALUE

Upon successful completion, *catopen*( ) shall return a message catalog descriptor for use on subsequent calls to *catgets*( ) and *catclose*( ). Otherwise, *catopen*( ) shall return (**nl_catd**) –1 and set *errno* to indicate the error.

ERRORS

The *catopen*( ) function may fail if:

[EACCES] Search permission is denied for the component of the path prefix of the message catalog or read permission is denied for the message catalog.

[EMFILE] All file descriptors available to the process are currently open.

[ENAMETOOLONG]
The length of a component of a pathname is longer than {NAME_MAX}.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

[ENFILE] Too many files are currently open in the system.

[ENOENT] The *name* argument contains a '/' and does not name an existing message catalog, the *name* argument does not contain a '/' and searching *NLSPATH* (if set) and then the implementation-defined default path for a message catalog

with that name failed, one or more files exist but all are of an invalid format, or the *name* argument points to an empty string.

[ENOMEM] Insufficient storage space is available.

[ENOTDIR] A component of the path prefix of the message catalog names an existing file that is neither a directory nor a symbolic link to a directory, or the pathname of the message catalog contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

EXAMPLES

None.

APPLICATION USAGE

Some implementations of *catopen*( ) use *malloc*( ) to allocate space for internal buffer areas. The *catopen*( ) function may fail if there is insufficient storage space available to accommodate these buffers.

Conforming applications must assume that message catalog descriptors are not valid after a call to one of the *exec* functions.

Application developers should be aware that guidelines for the location of message catalogs have not yet been developed. Therefore they should take care to avoid conflicting with catalogs used by other applications and the standard utilities.

To be sure that messages produced by an application running with appropriate privileges cannot be used by an attacker setting an unexpected value for *NLSPATH* in the environment to confuse a system administrator, such applications should use pathnames containing a `'/'` to get defined behavior when using *catopen*( ) to open a message catalog.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

catclose( ), *catgets*( )

XBD Chapter 8 (on page 167), **<fcntl.h>**, **<nl_types.h>**,

CHANGE HISTORY

First released in Issue 2.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

The *catopen*( ) function is moved from the XSI option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0045 [324] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0054 [645], XSH/TC2-2008/0055 [497], and XSH/TC2-2008/0056 [497] are applied.

Issue 8

Austin Group Defect 1122 is applied, clarifying that *catopen*( ) ignores files that are not valid message catalog files when performing an *NLSPATH* search.

Austin Group Defect 1516 is applied, adding XSI shading to text relating to *NLSPATH*.

NAME

cbrt, cbrtf, cbrtl — cube root functions

SYNOPSIS

```
#include <math.h>

double cbrt(double x);
float cbrtf(float x);
long double cbrtl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the real cube root of their argument *x*.

RETURN VALUE

Upon successful completion, these functions shall return the cube root of *x*.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

For some applications, a true cube root function, which returns negative results for negative arguments, is more appropriate than *pow*(*x*, 1.0/3.0), which returns a NaN for *x* less than 0.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD **<math.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The *cbrt*( ) function is no longer marked as an extension.

The *cbrtf*( ) and *cbrtl*( ) functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

NAME

ccos, ccosf, ccosl — complex cosine functions

SYNOPSIS

```
#include <complex.h>

double complex ccos(double complex z);
float complex ccosf(float complex z);
long double complex ccosl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the complex cosine of *z*.

RETURN VALUE

These functions shall return the complex cosine value.

MXC *ccos*(*conj*(*iz*)), *ccosf*(*conjf*(*iz*)), and *ccosl*(*conjl*(*iz*)) shall return exactly the same value as *conj*(*ccos*(*iz*)), *conjf*(*ccosf*(*iz*)), and *conjl*(*ccosl*(*iz*)), respectively, and *ccos*(–*iz*), *ccosf*(–*iz*), and *ccosl*(–*iz*) shall return exactly the same value as *ccos*(*iz*), *ccosf*(*iz*), and *ccosl*(*iz*), respectively, including for the special values of *iz* below.

If *iz* is +0 + *i*0, 1 + *i*0 shall be returned.

If *iz* is +0 + *i*Inf, NaN ± *i*0 shall be returned and the invalid floating-point exception shall be raised; the sign of the imaginary part of the result is unspecified.

If *iz* is +0 + *i*NaN, NaN ± *i*0 shall be returned; the sign of the imaginary part of the result is unspecified.

If *iz* is *x* + *i*Inf where *x* is non-zero and finite, NaN + *i*NaN shall be returned and the invalid floating-point exception shall be raised.

If *iz* is *x* + *i*NaN where *x* is non-zero and finite, NaN + *i*NaN shall be returned and the invalid floating-point exception may be raised.

If *iz* is +Inf + *i*0, +Inf + *i*0 shall be returned.

If *iz* is +Inf + *iy* where *y* is non-zero and finite, +Inf (*cos*(*y*) + *i sin*(*y*)) shall be returned.

If *iz* is +Inf + *i*Inf, ±Inf + *i*NaN shall be returned and the invalid floating-point exception shall be raised; the sign of the real part of the result is unspecified.

If *iz* is +Inf + *i*NaN, +Inf + *i*NaN shall be returned.

If *iz* is NaN + *i*0, NaN ± *i*0 shall be returned; the sign of the imaginary part of the result is unspecified.

If *iz* is NaN + *iy* where *y* is any non-zero number, NaN + *i*NaN shall be returned and the invalid floating-point exception may be raised.

If *iz* is NaN + *i*NaN, NaN + *i*NaN shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The MXC special cases for *ccos*( ) are derived from those for *ccosh*( ) by applying the formula *ccos*(*z*) = *ccosh*(*iz*).

FUTURE DIRECTIONS

None.

SEE ALSO

ccosh( ), *cacos*( )

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

NAME

ccosh, ccoshf, ccoshl — complex hyperbolic cosine functions

SYNOPSIS

```
#include <complex.h>

double complex ccosh(double complex z);
float complex ccoshf(float complex z);
long double complex ccoshl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the complex hyperbolic cosine of *z*.

RETURN VALUE

These functions shall return the complex hyperbolic cosine value.

MXC *ccosh*(*conj*(*z*)), *ccoshf*(*conjf*(*z*)), and *ccoshl*(*conjl*(*z*)) shall return exactly the same value as *conj*(*ccosh*(*z*)), *conjf*(*ccoshf*(*z*)), and *conjl*(*ccoshl*(*z*)), respectively, and *ccosh*(−*z*), *ccoshf*(−*z*), and *ccoshl*(−*z*) shall return exactly the same value as *ccosh*(*z*), *ccoshf*(*z*), and *ccoshl*(*z*), respectively, including for the special values of *z* below.

If *z* is +0 + *i*0, 1 + *i*0 shall be returned.

If *z* is +0 + *i*Inf, NaN ± *i*0 shall be returned and the invalid floating-point exception shall be raised; the sign of the imaginary part of the result is unspecified.

If *z* is +0 + *i*NaN, NaN ± *i*0 shall be returned; the sign of the imaginary part of the result is unspecified.

If *z* is *x* + *i*Inf where *x* is non-zero and finite, NaN + *i*NaN shall be returned and the invalid floating-point exception shall be raised.

If *z* is *x* + *i*NaN where *x* is non-zero and finite, NaN + *i*NaN shall be returned and the invalid floating-point exception may be raised.

If *z* is +Inf + *i*0, +Inf + *i*0 shall be returned.

If *z* is +Inf + *iy* where *y* is non-zero and finite, +Inf (*cos*(*y*) + *i sin*(*y*)) shall be returned.

If *z* is +Inf + *i*Inf, ±Inf + *i*NaN shall be returned and the invalid floating-point exception shall be raised; the sign of the real part of the result is unspecified.

If *z* is +Inf + *i*NaN, +Inf + *i*NaN shall be returned.

If *z* is NaN + *i*0, NaN ± *i*0 shall be returned; the sign of the imaginary part of the result is unspecified.

If *z* is NaN + *iy* where *y* is any non-zero number, NaN + *i*NaN shall be returned and the invalid floating-point exception may be raised.

If *z* is NaN + *i*NaN, NaN + *i*NaN shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cacosh( )

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

24562 **NAME**

24563 ccosl — complex cosine functions

24564 **SYNOPSIS**

24565
```
#include <complex.h>
```

24566
```
long double complex ccosl(long double complex z);
```

24567 **DESCRIPTION**

24568 Refer to *ccos( )*.

NAME

ceil, ceilf, ceill — ceiling value function

SYNOPSIS

```
#include <math.h>

double ceil(double x);
float ceilf(float x);
long double ceill(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the smallest integral value not less than *x*.

MX These functions may raise the inexact floating-point exception for finite non-integer arguments.

RETURN VALUE

MX The returned value shall be independent of the current rounding direction mode and shall have the same sign as *x*.

Upon successful completion, *ceil*( ), *ceilf*( ), and *ceill*( ) shall return the smallest integral value not less than *x*, expressed as a type **double**, **float**, or **long double**, respectively.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The integral value returned by these functions need not be expressible as an **intmax_t**. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *floor*( ), *isnan*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *ceilf*( ) and *ceill*( ) functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0046 [346] is applied.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

NAME

cexp, cexpf, cexpl — complex exponential functions

SYNOPSIS

```
#include <complex.h>

double complex cexp(double complex z);
float complex cexpf(float complex z);
long double complex cexpl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the complex exponent of z, defined as e^z.

RETURN VALUE

These functions shall return the complex exponential value of z.

MXC *cexp*(*conj*(*z*)), *cexpf* (*conjf* (*z*)), and *cexpl*(*conjl*(*z*)) shall return exactly the same value as *conj*(*cexp*(*z*)), *conjf* (*cexpf* (*z*)), and *conjl*(*cexpl*(*z*)), respectively, including for the special values of z below.

If z is $\pm 0 + i0$, $1 + i0$ shall be returned.

If z is $x + i\text{Inf}$ where x is finite, NaN + iNaN shall be returned and the invalid floating-point exception shall be raised.

If z is $x + i\text{NaN}$ where x is finite, NaN + iNaN shall be returned and the invalid floating-point exception may be raised.

If z is +Inf + $i0$, +Inf + $i0$ shall be returned.

If z is –Inf + iy where y is finite, $+0\ (cos(y) + i\,sin(y))$ shall be returned.

If z is +Inf + iy where y is non-zero and finite, $+\text{Inf}\ (cos(y) + i\,sin(y))$ shall be returned.

If z is –Inf + iInf, $\pm 0 \pm i0$ shall be returned; the signs of the real and imaginary parts of the result are unspecified.

If z is +Inf + iInf, ±Inf + iNaN shall be returned and the invalid floating-point exception shall be raised; the sign of the real part of the result is unspecified.

If z is –Inf + iNaN, $\pm 0 \pm i0$ shall be returned; the signs of the real and imaginary parts of the result are unspecified.

If z is +Inf + iNaN, ±Inf + iNaN shall be returned; the sign of the real part of the result is unspecified.

If z is NaN + $i0$, NaN + $i0$ shall be returned.

If z is NaN + iy where y is any non-zero number, NaN + iNaN shall be returned and the invalid floating-point exception may be raised.

If z is NaN + iNaN, NaN + iNaN shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

clog( )

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

NAME

cfgetispeed — get input baud rate

SYNOPSIS

```
#include <termios.h>

speed_t cfgetispeed(const struct termios *termios_p);
```

DESCRIPTION

The *cfgetispeed*( ) function shall extract the input baud rate from the **termios** structure to which the *termios_p* argument points.

This function shall return exactly the value in the **termios** data structure, without interpretation.

RETURN VALUE

Upon successful completion, *cfgetispeed*( ) shall return a value of type **speed_t** representing the input baud rate.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The term ``baud'' is used historically here, but is not technically correct. This is properly ``bits per second'', which may not be the same as baud. However, the term is used because of the historical usage and understanding.

The *cfgetospeed*( ), *cfgetispeed*( ), *cfsetospeed*( ), and *cfsetispeed*( ) functions do not take arguments as numbers, but rather as symbolic names. There are two reasons for this:

1. Historically, numbers were not used because of the way the rate was stored in the data structure. This is retained even though a function is now used.
2. More importantly, only a limited set of possible rates is at all portable, and this constrains the application to that set.

There is nothing to prevent an implementation accepting as an extension a number (such as 126), and since the encoding of the Bxxx symbols is not specified, this can be done to avoid introducing ambiguity.

Setting the input baud rate to zero was a mechanism to allow for split baud rates. Clarifications in this volume of POSIX.1-2024 have made it possible to determine whether split rates are supported and to support them without having to treat zero as a special case. Since this functionality is also confusing, it has been declared obsolescent. The 0 argument referred to is the literal constant 0, not the symbolic constant B0. This volume of POSIX.1-2024 does not preclude B0 from being defined as the value 0; in fact, implementations would likely benefit from the two being equivalent. This volume of POSIX.1-2024 does not fully specify whether the previous *cfsetispeed*( ) value is retained after a *tcgetattr*( ) as the actual value or as zero. Therefore, conforming applications should always set both the input speed and output speed when setting either.

In historical implementations, the baud rate information is traditionally kept in **c_cflag**. Applications should be written to presume that this might be the case (and thus not blindly copy **c_cflag**), but not to rely on it in case it is in some other field of the structure. Setting the **c_cflag** field absolutely after setting a baud rate is a non-portable action because of this. In general, the

unused parts of the flag fields might be used by the implementation and should not be blindly copied from the descriptions of one terminal device to another.

FUTURE DIRECTIONS

None.

SEE ALSO

cfgetospeed( ), *cfsetispeed*( ), *cfsetospeed*( ), *tcgetattr*( )

XBD Chapter 11 (on page 199), **<termios.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

NAME

cfgetospeed — get output baud rate

SYNOPSIS

```
#include <termios.h>

speed_t cfgetospeed(const struct termios *termios_p);
```

DESCRIPTION

The *cfgetospeed*( ) function shall extract the output baud rate from the **termios** structure to which the *termios_p* argument points.

This function shall return exactly the value in the **termios** data structure, without interpretation.

RETURN VALUE

Upon successful completion, *cfgetospeed*( ) shall return a value of type **speed_t** representing the output baud rate.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to *cfgetispeed*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

cfgetispeed( ), *cfsetispeed*( ), *cfsetospeed*( ), *tcgetattr*( )

XBD Chapter 11 (on page 199), **<termios.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

NAME

cfsetispeed — set input baud rate

SYNOPSIS

```
#include <termios.h>

int cfsetispeed(struct termios *termios_p, speed_t speed);
```

DESCRIPTION

The *cfsetispeed*( ) function shall set the input baud rate stored in the structure pointed to by *termios_p* to *speed*.

There shall be no effect on the baud rates set in the hardware until a subsequent successful call to *tcsetattr*( ) with the same **termios** structure. Similarly, errors resulting from attempts to set baud rates not supported by the terminal device need not be detected until the *tcsetattr*( ) function is called.

RETURN VALUE

Upon successful completion, *cfsetispeed*( ) shall return 0; otherwise, –1 shall be returned, and *errno* may be set to indicate the error.

ERRORS

The *cfsetispeed*( ) function may fail if:

[EINVAL] The *speed* value is not a valid baud rate.

[EINVAL] The value of *speed* is outside the range of possible speed values as specified in **<termios.h>**.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to *cfgetispeed*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

cfgetispeed( ), *cfgetospeed*( ), *cfsetospeed*( ), *tcsetattr*( )

XBD Chapter 11 (on page 199), **<termios.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The optional setting of *errno* and the [EINVAL] error conditions are added.

NAME

cfsetospeed — set output baud rate

SYNOPSIS

```
#include <termios.h>

int cfsetospeed(struct termios *termios_p, speed_t speed);
```

DESCRIPTION

The *cfsetospeed*( ) function shall set the output baud rate stored in the structure pointed to by *termios_p* to *speed*.

There shall be no effect on the baud rates set in the hardware until a subsequent successful call to *tcsetattr*( ) with the same **termios** structure. Similarly, errors resulting from attempts to set baud rates not supported by the terminal device need not be detected until the *tcsetattr*( ) function is called.

RETURN VALUE

Upon successful completion, *cfsetospeed*( ) shall return 0; otherwise, it shall return –1 and *errno* may be set to indicate the error.

ERRORS

The *cfsetospeed*( ) function may fail if:

[EINVAL] The *speed* value is not a valid baud rate.

[EINVAL] The value of *speed* is outside the range of possible speed values as specified in **<termios.h>**.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to *cfgetispeed*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

cfgetispeed( ), *cfgetospeed*( ), *cfsetispeed*( ), *tcsetattr*( )

XBD Chapter 11 (on page 199), **<termios.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The optional setting of *errno* and the [EINVAL] error conditions are added.

NAME

chdir — change working directory

SYNOPSIS

```
#include <unistd.h>

int chdir(const char *path);
```

DESCRIPTION

The *chdir*( ) function shall cause the directory named by the pathname pointed to by the *path* argument to become the current working directory; that is, the starting point for path searches for pathnames not beginning with '/'.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned, the current working directory shall remain unchanged, and *errno* shall be set to indicate the error.

ERRORS

The *chdir*( ) function shall fail if:

[EACCES] Search permission is denied for any component of the pathname.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of a component of a pathname is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing directory or *path* is an empty string.

[ENOTDIR] A component of the pathname names an existing file that is neither a directory nor a symbolic link to a directory.

The *chdir*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

EXAMPLES

Changing the Current Working Directory

The following example makes the value pointed to by **directory**, **/tmp**, the current working directory.

```
#include <unistd.h>
...
char *directory = "/tmp";
int ret;

ret = chdir (directory);
```


APPLICATION USAGE

None.

RATIONALE

The *chdir*( ) function only affects the working directory of the current process.

FUTURE DIRECTIONS

None.

SEE ALSO

getcwd( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The APPLICATION USAGE section is added.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0047 [324] is applied.

NAME

chmod, fchmodat — change mode of a file

SYNOPSIS

```
#include <sys/stat.h>

int chmod(const char *path, mode_t mode);
```

OH
```
#include <fcntl.h>

int fchmodat(int fd, const char *path, mode_t mode, int flag);
```

DESCRIPTION

XSI The *chmod*( ) function shall change S_ISUID, S_ISGID, S_ISVTX, and the file permission bits of the file named by the pathname pointed to by the *path* argument to the corresponding bits in the *mode* argument. If any bits that can be set in the *st_mode* value returned by *lstat*( ) or *stat*( ) but cannot be changed using *chmod*( ), such as the bits that are used to encode the file type, are set in the *mode* argument, these *read-only st_mode bits* shall be ignored.

If the effective user ID of the process does not match the owner of the file and the process does not have appropriate privileges, the *chmod*( ) function shall fail.

XSI S_ISUID, S_ISGID, S_ISVTX, and the file permission bits are described in **<sys/stat.h>**.

If the calling process does not have appropriate privileges, and if the group ID of the file does not match the effective group ID or one of the supplementary group IDs and if the file is a regular file, bit S_ISGID (set-group-ID on execution) in the file's mode shall be cleared upon successful return from *chmod*( ).

Additional implementation-defined restrictions may cause the S_ISUID and S_ISGID bits in
XSI *mode* to be ignored, and may cause the S_ISVTX bit in *mode* to be ignored for non-directory files.

Upon successful completion, *chmod*( ) shall mark for update the last file status change timestamp of the file.

The *fchmodat*( ) function shall be equivalent to the *chmod*( ) function except in the case where *path* specifies a relative path. In this case the file to be changed is determined relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in **<fcntl.h>**:

AT_SYMLINK_NOFOLLOW
: If *path* names a symbolic link, then the mode of the symbolic link is changed.

If *fchmodat*( ) is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used. If also *flag* is zero, the behavior shall be identical to a call to *chmod*( ).

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return –1 and set *errno* to indicate the error. If –1 is returned, no change to the file mode occurs.

24931 **ERRORS**

24932 These functions shall fail if:

24933 [EACCES] Search permission is denied on a component of the path prefix.

24934 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
24935 argument.

24936 [ENAMETOOLONG]
24937 The length of a component of a pathname is longer than {NAME_MAX}.

24938 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

24939 [ENOTDIR] A component of the path prefix names an existing file that is neither a
24940 directory nor a symbolic link to a directory, or the *path* argument contains at
24941 least one non-<slash> character and ends with one or more trailing <slash>
24942 characters and the last pathname component names an existing file that is
24943 neither a directory nor a symbolic link to a directory.

24944 [EPERM] The effective user ID does not match the owner of the file and the process does
24945 not have appropriate privileges.

24946 [EROFS] The named file resides on a read-only file system.

24947 The *fchmodat*( ) function shall fail if:

24948 [EACCES] The access mode of the open file description associated with *fd* is not
24949 O_SEARCH and the permissions of the directory underlying *fd* do not permit
24950 directory searches.

24951 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
24952 neither AT_FDCWD nor a valid file descriptor open for reading or searching.

24953 [ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated
24954 with a non-directory file.

24955 These functions may fail if:

24956 [EINTR] A signal was caught during execution of the function.

24957 [EINVAL] The value of the *mode* argument, ignoring *read-only st_mode bits* (see the
24958 DESCRIPTION), is invalid.

24959 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
24960 resolution of the *path* argument.

24961 [ENAMETOOLONG]
24962 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
24963 symbolic link produced an intermediate result with a length that exceeds
24964 {PATH_MAX}.

24965 The *fchmodat*( ) function may fail if:

24966 [EINVAL] The value of the *flag* argument is invalid.

24967 [EOPNOTSUPP] The AT_SYMLINK_NOFOLLOW bit is set in the *flag* argument, *path* names a
24968 symbolic link, and the system does not support changing the mode of a
24969 symbolic link.

EXAMPLES

Setting Read Permissions for User, Group, and Others

The following example sets read permissions for the owner, group, and others.

```
#include <sys/stat.h>

const char *path;
...
chmod(path, S_IRUSR|S_IRGRP|S_IROTH);
```

Setting Read, Write, and Execute Permissions for the Owner Only

The following example sets read, write, and execute permissions for the owner, and no permissions for group and others.

```
#include <sys/stat.h>

const char *path;
...
chmod(path, S_IRWXU);
```

Setting Different Permissions for Owner, Group, and Other

The following example sets owner permissions for CHANGEFILE to read, write, and execute, group permissions to read and execute, and other permissions to read.

```
#include <sys/stat.h>

#define CHANGEFILE "/etc/myfile"
...
chmod(CHANGEFILE, S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH);
```

Modifying File Permissions

The following example adds group write permission to the existing permission bits for a file if that bit is not already set.

```
#include <sys/stat.h>

struct stat sbuf;
...
if (stat(path, &sbuf) == 0 && (sbuf.st_mode & S_IWGRP) == 0)
    chmod(path, sbuf.st_mode | S_IWGRP);
```

Setting and Checking File Permissions

The following example sets the file permission bits for a file named **/home/cnd/mod1**, then calls the *stat*( ) function to verify the permissions.

```
#include <sys/types.h>
#include <sys/stat.h>

int status;
struct stat buffer
...
```



```
chmod("/home/cnd/mod1", S_IRWXU|S_IRWXG|S_IROTH|S_IWOTH);
status = stat("/home/cnd/mod1", &buffer);
```

APPLICATION USAGE

In order to ensure that the S_ISUID and S_ISGID bits are set, an application requiring this should use *stat*( ) after a successful *chmod*( ) to verify this.

Any file descriptors currently open by any process on the file could possibly become invalid if the mode of the file is changed to a value which would deny access to that process. One situation where this could occur is on a stateless file system. This behavior will not occur in a conforming environment.

RATIONALE

This volume of POSIX.1-2024 specifies that the S_ISGID bit is cleared by *chmod*( ) on a regular file under certain conditions. This is specified on the assumption that regular files may be executed, and the system should prevent users from making executable *setgid*( ) files perform with privileges that the caller does not have. On implementations that support execution of other file types, the S_ISGID bit should be cleared for those file types under the same circumstances.

Implementations that use the S_ISUID bit to indicate some other function (for example, mandatory record locking) on non-executable files need not clear this bit on writing. They should clear the bit for executable files and any other cases where the bit grants special powers to processes that change the file contents. Similar comments apply to the S_ISGID bit.

The purpose of the *fchmodat*( ) function is to enable changing the mode of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *chmod*( ), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *fchmodat*( ) function it can be guaranteed that the changed file is located relative to the desired directory. Some implementations might allow changing the mode of symbolic links. This is not supported by the interfaces in the POSIX specification. Systems with such support provide an interface named *lchmod*( ). To support such implementations *fchmodat*( ) has a *flag* parameter.

FUTURE DIRECTIONS

None.

SEE ALSO

access( ), *chown*( ), *exec*, *fstatat*( ), *fstatvfs*( ), *mkdir*( ), *mkfifo*( ), *mknod*( ), *open*( )

XBD **<fcntl.h>**, **<sys/stat.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [EINVAL] and [EINTR] optional error conditions are added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

The *fchmodat*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

Changes are made to allow a directory to be opened for searching.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0048 [300], XSH/TC1-2008/0049 [461], XSH/TC1-2008/0050 [324], XSH/TC1-2008/0051 [278], and XSH/TC1-2008/0052 [278] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0057 [873], XSH/TC2-2008/0058 [591], XSH/TC2-2008/0059 [817], XSH/TC2-2008/0060 [817], and XSH/TC2-2008/0061 [893] are applied.

Issue 8

Austin Group Defect 1024 is applied, allowing the S_ISVTX bit to be ignored for non-directory files.

Austin Group Defect 1283 is applied, clarifying that *chmod*( ) ignores *read-only st_mode bits* in the *mode* argument.

25071 **NAME**

25072 chown, fchownat — change owner and group of a file

25073 **SYNOPSIS**

25074 ```
#include <unistd.h>
```

25075 ```
int chown(const char *path, uid_t owner, gid_t group);
```

25076 OH ```
#include <fcntl.h>
```

25077 ```
int fchownat(int fd, const char *path, uid_t owner, gid_t group,
25078     int flag);
```

25079 **DESCRIPTION**

25080 The *chown*( ) function shall change the user and group ownership of a file.

25081 The *path* argument points to a pathname naming a file. The user ID and group ID of the named
25082 file shall be set to the numeric values contained in *owner* and *group*, respectively.

25083 Only processes with an effective user ID equal to the user ID of the file or with appropriate
25084 privileges may change the ownership of a file. If _POSIX_CHOWN_RESTRICTED is in effect for
25085 *path*:

25086 • Changing the user ID is restricted to processes with appropriate privileges.

25087 • Changing the group ID is permitted to a process with an effective user ID equal to the user
25088 ID of the file, but without appropriate privileges, if and only if *owner* is equal to the file's
25089 user ID or (**uid_t**)–1 and *group* is equal either to the calling process' effective group ID or to
25090 one of its supplementary group IDs.

25091 If the specified file is a regular file, one or more of the S_IXUSR, S_IXGRP, or S_IXOTH bits of
25092 the file mode are set, and the process does not have appropriate privileges, the set-user-ID
25093 (S_ISUID) and set-group-ID (S_ISGID) bits of the file mode shall be cleared upon successful
25094 return from *chown*( ). If the specified file is a regular file, one or more of the S_IXUSR, S_IXGRP,
25095 or S_IXOTH bits of the file mode are set, and the process has appropriate privileges, it is
25096 implementation-defined whether the set-user-ID and set-group-ID bits are altered. If the *chown*( )
25097 function is successfully invoked on a file that is not a regular file and one or more of the
25098 S_IXUSR, S_IXGRP, or S_IXOTH bits of the file mode are set, the set-user-ID and set-group-ID
25099 bits may be cleared.

25100 If *owner* or *group* is specified as (**uid_t**)–1 or (**gid_t**)–1, respectively, the corresponding ID of the
25101 file shall not be changed.

25102 Upon successful completion, *chown*( ) shall mark for update the last file status change timestamp
25103 of the file, except that if *owner* is (**uid_t**)–1 and *group* is (**gid_t**)–1, the file status change
25104 timestamp need not be marked for update.

25105 The *fchownat*( ) function shall be equivalent to the *chown*( ) and *lchown*( ) functions except in the
25106 case where *path* specifies a relative path. In this case the file to be changed is determined relative
25107 to the directory associated with the file descriptor *fd* instead of the current working directory. If
25108 the access mode of the open file description associated with the file descriptor is not
25109 O_SEARCH, the function shall check whether directory searches are permitted using the current
25110 permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the
25111 function shall not perform the check.

25112 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined
25113 in **<fcntl.h>**:

AT_SYMLINK_NOFOLLOW
If *path* names a symbolic link, ownership of the symbolic link is changed.

If *fchownat*( ) is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used and the behavior shall be identical to a call to *chown*( ) or *lchown*( ) respectively, depending on whether or not the AT_SYMLINK_NOFOLLOW bit is set in the *flag* argument.

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return –1 and set *errno* to indicate the error. If –1 is returned, no changes are made in the user ID and group ID of the file.

ERRORS

These functions shall fail if:

[EACCES] Search permission is denied on a component of the path prefix.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of a component of a pathname is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

[EPERM] The effective user ID does not match the owner of the file, or the calling process does not have appropriate privileges and _POSIX_CHOWN_RESTRICTED indicates that such privilege is required.

[EROFS] The named file resides on a read-only file system.

The *fchownat*( ) function shall fail if:

[EACCES] The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

[EBADF] The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

[ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

These functions may fail if:

[EIO] An I/O error occurred while reading or writing to the file system.

[EINTR] The *chown*( ) function was interrupted by a signal which was caught.

[EINVAL] The owner or group ID supplied is not a value supported by the implementation.

25154 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
25155 resolution of the *path* argument.

25156 [ENAMETOOLONG]
25157 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
25158 symbolic link produced an intermediate result with a length that exceeds
25159 {PATH_MAX}.

25160 The *fchownat*( ) function may fail if:

25161 [EINVAL] The value of the *flag* argument is not valid.

25162 **EXAMPLES**

25163 None.

25164 **APPLICATION USAGE**

25165 Although *chown*( ) can be used on some implementations by the file owner to change the owner
25166 and group to any desired values, the only portable use of this function is to change the group of
25167 a file to the effective GID of the calling process or to a member of its group set.

25168 **RATIONALE**

25169 System III and System V allow a user to give away files; that is, the owner of a file may change
25170 its user ID to anything. This is a serious problem for implementations that are intended to meet
25171 government security regulations. Version 7 and 4.3 BSD permit only the superuser to change the
25172 user ID of a file. Some government agencies (usually not ones concerned directly with security)
25173 find this limitation too confining. This volume of POSIX.1-2024 uses *may* to permit secure
25174 implementations while not disallowing System V.

25175 System III and System V allow the owner of a file to change the group ID to anything. Version 7
25176 permits only the superuser to change the group ID of a file. 4.3 BSD permits the owner to
25177 change the group ID of a file to its effective group ID or to any of the groups in the list of
25178 supplementary group IDs, but to no others.

25179 The POSIX.1-1990 standard requires that the *chown*( ) function invoked by a non-appropriate
25180 privileged process clear the S_ISGID and the S_ISUID bits for regular files, and permits them to
25181 be cleared for other types of files. This is so that changes in accessibility do not accidentally
25182 cause files to become security holes. Unfortunately, requiring these bits to be cleared on non-
25183 executable data files also clears the mandatory file locking bit (shared with S_ISGID), which is
25184 an extension on many implementations (it first appeared in System V). These bits should only be
25185 required to be cleared on regular files that have one or more of their execute bits set.

25186 The purpose of the *fchownat*( ) function is to enable changing ownership of files in directories
25187 other than the current working directory without exposure to race conditions. Any part of the
25188 path of a file could be changed in parallel to a call to *chown*( ) or *lchown*( ), resulting in
25189 unspecified behavior. By opening a file descriptor for the target directory and using the
25190 *fchownat*( ) function it can be guaranteed that the changed file is located relative to the desired
25191 directory.

25192 **FUTURE DIRECTIONS**

25193 None.

25194 **SEE ALSO**

25195 *chmod*( ), *fpathconf*( ), *lchown*( )

25196 XBD **<fcntl.h>**, **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- The wording describing the optional dependency on _POSIX_CHOWN_RESTRICTED is restored.
- The [EPERM] error is restored as an error dependent on _POSIX_CHOWN_RESTRICTED. This is since its operand is a pathname and applications should be aware that the error may not occur for that pathname if the file system does not support _POSIX_CHOWN_RESTRICTED.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The value for *owner* of (**uid_t**)–1 allows the use of –1 by the owner of a file to change the group ID only. A corresponding change is made for group.
- The [ELOOP] mandatory error condition is added.
- The [EIO] and [EINTR] optional error conditions are added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Clarification is added that the S_ISUID and S_ISGID bits do not need to be cleared when the process has appropriate privileges.
- The [ELOOP] optional error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

The *fchownat*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

Changes are made to allow a directory to be opened for searching.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0053 [461], XSH/TC1-2008/0054 [324], XSH/TC1-2008/0055 [278], and XSH/TC1-2008/0056 [278] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0062 [873], XSH/TC2-2008/0063 [591], XSH/TC2-2008/0064 [485], XSH/TC2-2008/0065 [817], and XSH/TC2-2008/0066 [817] are applied.

NAME

cimag, cimagf, cimagl — complex imaginary functions

SYNOPSIS

```
#include <complex.h>

double cimag(double complex z);
float cimagf(float complex z);
long double cimagl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the imaginary part of *z*.

RETURN VALUE

These functions shall return the imaginary part value (as a real).

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

For a variable *z* of complex type:

```
z == creal(z) + cimag(z)*I
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

carg( ), *conj*( ), *cproj*( ), *creal*( )

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

25264 **NAME**

25265 clearerr — clear indicators on a stream

25266 **SYNOPSIS**

25267 `#include <stdio.h>`

25268 `void clearerr(FILE *stream);`

25269 **DESCRIPTION**

25270 CX The functionality described on this reference page is aligned with the ISO C standard. Any
25271 conflict between the requirements described here and the ISO C standard is unintentional. This
25272 volume of POSIX.1-2024 defers to the ISO C standard.

25273 The *clearerr*( ) function shall clear the end-of-file and error indicators for the stream to which
25274 *stream* points.

25275 CX The *clearerr*( ) function shall not change the setting of *errno* if *stream* is valid.

25276 **RETURN VALUE**

25277 The *clearerr*( ) function shall not return a value.

25278 **ERRORS**

25279 No errors are defined.

25280 **EXAMPLES**

25281 None.

25282 **APPLICATION USAGE**

25283 None.

25284 **RATIONALE**

25285 None.

25286 **FUTURE DIRECTIONS**

25287 None.

25288 **SEE ALSO**

25289 XBD **<stdio.h>**

25290 **CHANGE HISTORY**

25291 First released in Issue 1. Derived from Issue 1 of the SVID.

25292 **Issue 7**

25293 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0057 [401] is applied.

NAME

clock — report CPU time used

SYNOPSIS

```
#include <time.h>

clock_t clock(void);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *clock*( ) function shall return the implementation's best approximation to the processor time used by the process since the beginning of an implementation-defined era related only to the process invocation.

RETURN VALUE

To determine the time in seconds, the value returned by *clock*( ) should be divided by the value XSI of the macro CLOCKS_PER_SEC. CLOCKS_PER_SEC is defined to be one million in **<time.h>**. If the processor time used is not available or its value cannot be represented, the function shall return the value (**clock_t**)–1.

ERRORS

The *clock*( ) function shall fail if:

CX [EOVERFLOW] The processor time used cannot be represented in an object of type **clock_t**.

EXAMPLES

None.

APPLICATION USAGE

In programming environments where **clock_t** is a 32-bit integer type and CLOCKS_PER_SEC is one million, *clock*( ) will start failing in less than 36 minutes of processor time for signed **clock_t**, or 72 minutes for unsigned **clock_t**. Applications intended to be portable to such environments should use *times*( ) instead (or *clock_gettime*( ) with CLOCK_PROCESS_CPUTIME_ID, if supported).

In order to measure the time spent in a program, *clock*( ) should be called at the start of the program and its return value subtracted from the value returned by subsequent calls. The value returned by *clock*( ) is defined for compatibility across systems that have clocks with different resolutions. The resolution on any particular system need not be to microsecond accuracy.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime( ), *clock_getres*( ), *ctime*( ), *difftime*( ), *futimens*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( ), *times*( )

XBD **<time.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0067 [686] is applied.

Issue 8

Austin Group Defect 703 is applied, adding the [EOVERFLOW] error.

25340 **NAME**

25341 clock_getcpuclockid — access a process CPU-time clock (**ADVANCED REALTIME**)

25342 **SYNOPSIS**

25343 CPT
```
#include <time.h>
```

25344
```
int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);
```

25345 **DESCRIPTION**

25346 The *clock_getcpuclockid*( ) function shall return the clock ID of the CPU-time clock of the process
25347 specified by *pid*. If the process described by *pid* exists and the calling process has permission, the
25348 clock ID of this clock shall be returned in *clock_id*.

25349 If *pid* is zero, the *clock_getcpuclockid*( ) function shall return the clock ID of the CPU-time clock of
25350 the process making the call, in *clock_id*.

25351 The conditions under which one process has permission to obtain the CPU-time clock ID of
25352 other processes are implementation-defined.

25353 **RETURN VALUE**

25354 Upon successful completion, *clock_getcpuclockid*( ) shall return zero; otherwise, an error number
25355 shall be returned to indicate the error.

25356 **ERRORS**

25357 The *clock_getcpuclockid*( ) function shall fail if:

25358 [EPERM] The requesting process does not have permission to access the CPU-time clock
25359 for the process.

25360 The *clock_getcpuclockid*( ) function may fail if:

25361 [ESRCH] No process can be found corresponding to the process specified by *pid*.

25362 **EXAMPLES**

25363 None.

25364 **APPLICATION USAGE**

25365 The *clock_getcpuclockid*( ) function is part of the Process CPU-Time Clocks option and need not be
25366 provided on all implementations.

25367 **RATIONALE**

25368 None.

25369 **FUTURE DIRECTIONS**

25370 None.

25371 **SEE ALSO**

25372 *clock_getres*( ), *timer_create*( )

25373 XBD **<time.h>**

25374 **CHANGE HISTORY**

25375 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

25376 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

NAME

clock_getres, clock_gettime, clock_settime — clock and timer functions

SYNOPSIS

CX
```
#include <time.h>

int clock_getres(clockid_t clock_id, struct timespec *res);
int clock_gettime(clockid_t clock_id, struct timespec *tp);
int clock_settime(clockid_t clock_id, const struct timespec *tp);
```

DESCRIPTION

The *clock_getres*( ) function shall return the resolution of any clock. Clock resolutions are implementation-defined and cannot be set by a process. If the argument *res* is not NULL, the resolution of the specified clock shall be stored in the location pointed to by *res*. If *res* is NULL, the clock resolution is not returned. If the *time* argument of *clock_settime*( ) is not a multiple of *res*, then the value is truncated to a multiple of *res*.

The *clock_gettime*( ) function shall return the current value *tp* for the specified clock, *clock_id*.

The *clock_settime*( ) function shall set the specified clock, *clock_id*, to the value specified by *tp*. Time values that are between two consecutive non-negative integer multiples of the resolution of the specified clock shall be truncated down to the smaller multiple of the resolution.

A clock may be system-wide (that is, visible to all processes) or per-process (measuring time that is meaningful only within a process). All implementations shall support a *clock_id* of CLOCK_REALTIME as defined in **<time.h>**. This clock represents the clock measuring real time for the system. For this clock, the values returned by *clock_gettime*( ) and specified by *clock_settime*( ) represent the amount of time (in seconds and nanoseconds) since the Epoch. An implementation may also support additional clocks. The interpretation of time values for these clocks is unspecified.

If the value of the CLOCK_REALTIME clock is set via *clock_settime*( ), the new value of the clock shall be used to determine the time of expiration for absolute time services based upon the CLOCK_REALTIME clock. This applies to the time at which armed absolute timers expire. If the absolute time requested at the invocation of such a time service is before the new value of the clock, the time service shall expire immediately as if the clock had reached the requested time normally.

Setting the value of the CLOCK_REALTIME clock via *clock_settime*( ) shall have no effect on threads that are blocked waiting for a relative time service based upon this clock, including the *nanosleep*( ) and *thrd_sleep*( ) functions; nor on the expiration of relative timers based upon this clock. Consequently, these time services shall expire when the requested relative interval elapses, independently of the new or old value of the clock.

All implementations shall support a *clock_id* of CLOCK_MONOTONIC defined in **<time.h>**. This clock represents the monotonic clock for the system. For this clock, the value returned by *clock_gettime*( ) represents the amount of time (in seconds and nanoseconds) since an unspecified point in the past (for example, system start-up time, or the Epoch). This point does not change after system start-up time. The value of the CLOCK_MONOTONIC clock cannot be set via *clock_settime*( ). This function shall fail if it is invoked with a *clock_id* argument of CLOCK_MONOTONIC.

The effect of setting a clock via *clock_settime*( ) on armed per-process timers associated with a clock other than CLOCK_REALTIME is implementation-defined.

If the value of the CLOCK_REALTIME clock is set via *clock_settime*( ), the new value of the clock shall be used to determine the time at which the system shall awaken a thread blocked on an

absolute *clock_nanosleep*( ) call based upon the CLOCK_REALTIME clock. If the absolute time requested at the invocation of such a time service is before the new value of the clock, the call shall return immediately as if the clock had reached the requested time normally.

Setting the value of the CLOCK_REALTIME clock via *clock_settime*( ) shall have no effect on any thread that is blocked on a relative *clock_nanosleep*( ) call. Consequently, the call shall return when the requested relative interval elapses, independently of the new or old value of the clock.

Appropriate privileges to set a particular clock are implementation-defined.

CPT If _POSIX_CPUTIME is defined, implementations shall support clock ID values obtained by invoking *clock_getcpuclockid*( ), which represent the CPU-time clock of a given process. Implementations shall also support the special **clockid_t** value CLOCK_PROCESS_CPUTIME_ID, which represents the CPU-time clock of the calling process when invoking one of the *clock_**( ) or *timer_**( ) functions. For these clock IDs, the values returned by *clock_gettime*( ) and specified by *clock_settime*( ) represent the amount of execution time of the process associated with the clock. Changing the value of a CPU-time clock via *clock_settime*( ) shall have no effect on the behavior of the sporadic server scheduling policy (see Scheduling Policies).

TCT If _POSIX_THREAD_CPUTIME is defined, implementations shall support clock ID values obtained by invoking *pthread_getcpuclockid*( ), which represent the CPU-time clock of a given thread. Implementations shall also support the special **clockid_t** value CLOCK_THREAD_CPUTIME_ID, which represents the CPU-time clock of the calling thread when invoking one of the *clock_**( ) or *timer_**( ) functions. For these clock IDs, the values returned by *clock_gettime*( ) and specified by *clock_settime*( ) shall represent the amount of execution time of the thread associated with the clock. Changing the value of a CPU-time clock via *clock_settime*( ) shall have no effect on the behavior of the sporadic server scheduling policy (see Scheduling Policies).

RETURN VALUE

A return value of 0 shall indicate that the call succeeded. A return value of –1 shall indicate that an error occurred, and *errno* shall be set to indicate the error.

ERRORS

The *clock_getres*( ), *clock_gettime*( ), and *clock_settime*( ) functions shall fail if:

[EINVAL] The *clock_id* argument does not specify a known clock.

The *clock_gettime*( ) function shall fail if:

[EOVERFLOW] The number of seconds will not fit in an object of type **time_t**.

The *clock_settime*( ) function shall fail if:

[EINVAL] The *tp* argument to *clock_settime*( ) is outside the range for the given clock ID.

[EINVAL] The *tp* argument specified a nanosecond value less than zero or greater than or equal to 1 000 million.

[EINVAL] The value of the *clock_id* argument is CLOCK_MONOTONIC.

The *clock_settime*( ) function may fail if:

[EPERM] The requesting process does not have appropriate privileges to set the specified clock.

EXAMPLES

None.

APPLICATION USAGE

Note that the absolute value of the monotonic clock is meaningless (because its origin is arbitrary), and thus there is no need to set it. Furthermore, realtime applications can rely on the fact that the value of this clock is never set and, therefore, that time intervals measured with this clock will not be affected by calls to *clock_settime*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Scheduling Policies (on page 531), *clock_getcpuclockid*( ), *clock_nanosleep*( ), *ctime*( ), *mq_receive*( ), *mq_send*( ), *nanosleep*( ), *pthread_mutex_clocklock*( ), *sem_clockwait*( ), *thrd_sleep*( ), *time*( ), *timer_create*( ), *timer_getoverrun*( )

XBD **<time.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Timers option.

The APPLICATION USAGE section is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Clarification is added of the effect of resetting the clock resolution.

CPU-time clocks and the *clock_getcpuclockid*( ) function are added for alignment with IEEE Std 1003.1d-1999.

The following changes are added for alignment with IEEE Std 1003.1j-2000:

- The DESCRIPTION is updated as follows:
  - — The value returned by *clock_gettime*( ) for CLOCK_MONOTONIC is specified.
  - — The *clock_settime*( ) function failing for CLOCK_MONOTONIC is specified.
  - — The effects of *clock_settime*( ) on the *clock_nanosleep*( ) function with respect to CLOCK_REALTIME are specified.
- An [EINVAL] error is added to the ERRORS section, indicating that *clock_settime*( ) fails for CLOCK_MONOTONIC.
- The APPLICATION USAGE section notes that the CLOCK_MONOTONIC clock need not and shall not be set by *clock_settime*( ) since the absolute value of the CLOCK_MONOTONIC clock is meaningless.
- The *clock_nanosleep*( ), *mq_timedreceive*( ), *mq_timedsend*( ), *pthread_mutex_timedlock*( ), *sem_timedwait*( ), *timer_create*( ), and *timer_settime*( ) functions are added to the SEE ALSO section.

25504 **Issue 7**

25505 Functionality relating to the Clock Selection option is moved to the Base.

25506 The *clock_getres*( ), *clock_gettime*( ), and *clock_settime*( ) functions are moved from the Timers
25507 option to the Base.

25508 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0058 [106] is applied.

25509 **Issue 8**

25510 Austin Group Defect 1302 is applied, changing ``the *nanosleep*( ) function'' to ``the *nanosleep*( ) and
25511 *thrd_sleep*( ) functions''.

25512 Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.

25513 **NAME**

25514 clock_nanosleep — high resolution sleep with specifiable clock

25515 **SYNOPSIS**

25516 CX
```
#include <time.h>

int clock_nanosleep(clockid_t clock_id, int flags,
    const struct timespec *rqtp, struct timespec *rmtp);
```

25519 **DESCRIPTION**

25520 If the flag TIMER_ABSTIME is not set in the *flags* argument, the *clock_nanosleep*( ) function shall
25521 cause the current thread to be suspended from execution until either the time interval specified
25522 by the *rqtp* argument has elapsed, or a signal is delivered to the calling thread and its action is to
25523 invoke a signal-catching function, or the process is terminated. The clock used to measure the
25524 time shall be the clock specified by *clock_id*.

25525 If the flag TIMER_ABSTIME is set in the *flags* argument, the *clock_nanosleep*( ) function shall
25526 cause the current thread to be suspended from execution until either the time value of the clock
25527 specified by *clock_id* reaches the absolute time specified by the *rqtp* argument, or a signal is
25528 delivered to the calling thread and its action is to invoke a signal-catching function, or the
25529 process is terminated. If, at the time of the call, the time value specified by *rqtp* is less than or
25530 equal to the time value of the specified clock, then *clock_nanosleep*( ) shall return immediately
25531 and the calling process shall not be suspended.

25532 The suspension time caused by this function may be longer than requested because the
25533 argument value is rounded up to an integer multiple of the sleep resolution, or because of the
25534 scheduling of other activity by the system. But, except for the case of being interrupted by a
25535 signal, the suspension time for the relative *clock_nanosleep*( ) function (that is, with the
25536 TIMER_ABSTIME flag not set) shall not be less than the time interval specified by *rqtp*, as
25537 measured by the corresponding clock. The suspension for the absolute *clock_nanosleep*( ) function
25538 (that is, with the TIMER_ABSTIME flag set) shall be in effect at least until the value of the
25539 corresponding clock reaches the absolute time specified by *rqtp*, except for the case of being
25540 interrupted by a signal.

25541 The use of the *clock_nanosleep*( ) function shall have no effect on the action or blockage of any
25542 signal.

25543 The *clock_nanosleep*( ) function shall fail if the *clock_id* argument refers to the CPU-time clock of
25544 the calling thread. It is unspecified whether *clock_id* values of other CPU-time clocks are allowed.

25545 **RETURN VALUE**

25546 If the *clock_nanosleep*( ) function returns because the requested time has elapsed, its return value
25547 shall be zero.

25548 If the *clock_nanosleep*( ) function returns because it has been interrupted by a signal, it shall
25549 return the corresponding error value. For the relative *clock_nanosleep*( ) function, if the *rmtp*
25550 argument is non-NULL, the **timespec** structure referenced by it shall be updated to contain the
25551 amount of time remaining in the interval (the requested time minus the time actually slept). The
25552 *rqtp* and *rmtp* arguments can point to the same object. If the *rmtp* argument is NULL, the
25553 remaining time is not returned. The absolute *clock_nanosleep*( ) function has no effect on the
25554 structure referenced by *rmtp*.

25555 If *clock_nanosleep*( ) fails, it shall return the corresponding error value.

ERRORS

The *clock_nanosleep*( ) function shall fail if:

[EINTR] The *clock_nanosleep*( ) function was interrupted by a signal.

[EINVAL] The *rqtp* argument specified a nanosecond value less than zero or greater than or equal to 1 000 million; or the TIMER_ABSTIME flag was specified in flags and the *rqtp* argument is outside the range for the clock specified by *clock_id*; or the *clock_id* argument does not specify a known clock, or specifies the CPU-time clock of the calling thread.

[ENOTSUP] The *clock_id* argument specifies a clock for which *clock_nanosleep*( ) is not supported, such as a CPU-time clock.

EXAMPLES

None.

APPLICATION USAGE

Calling *clock_nanosleep*( ) with the value TIMER_ABSTIME not set in the *flags* argument and with a *clock_id* of CLOCK_REALTIME is equivalent to calling *nanosleep*( ) with the same *rqtp* and *rmtp* arguments.

RATIONALE

The *nanosleep*( ) function specifies that the system-wide clock CLOCK_REALTIME is used to measure the elapsed time for this time service. However, with the introduction of the monotonic clock CLOCK_MONOTONIC a new relative sleep function is needed to allow an application to take advantage of the special characteristics of this clock.

There are many applications in which a process needs to be suspended and then activated multiple times in a periodic way; for example, to poll the status of a non-interrupting device or to refresh a display device. For these cases, it is known that precise periodic activation cannot be achieved with a relative *sleep*( ) or *nanosleep*( ) function call. Suppose, for example, a periodic process that is activated at time $T0$, executes for a while, and then wants to suspend itself until time $T0+T$, the period being T. If this process wants to use the *nanosleep*( ) function, it must first call *clock_gettime*( ) to get the current time, then calculate the difference between the current time and $T0+T$ and, finally, call *nanosleep*( ) using the computed interval. However, the process could be preempted by a different process between the two function calls, and in this case the interval computed would be wrong; the process would wake up later than desired. This problem would not occur with the absolute *clock_nanosleep*( ) function, since only one function call would be necessary to suspend the process until the desired time. In other cases, however, a relative sleep is needed, and that is why both functionalities are required.

Although it is possible to implement periodic processes using the timers interface, this implementation would require the use of signals, and the reservation of some signal numbers. In this regard, the reasons for including an absolute version of the *clock_nanosleep*( ) function in POSIX.1-2024 are the same as for the inclusion of the relative *nanosleep*( ).

It is also possible to implement precise periodic processes using *pthread_cond_timedwait*( ), in which an absolute timeout is specified that takes effect if the condition variable involved is never signaled. However, the use of this interface is unnatural, and involves performing other operations on mutexes and condition variables that imply an unnecessary overhead. Furthermore, *pthread_cond_timedwait*( ) is not available in implementations that do not support threads.

Although the interface of the relative and absolute versions of the new high resolution sleep service is the same *clock_nanosleep*( ) function, the *rmtp* argument is only used in the relative sleep. This argument is needed in the relative *clock_nanosleep*( ) function to reissue the function

call if it is interrupted by a signal, but it is not needed in the absolute *clock_nanosleep*( ) function call; if the call is interrupted by a signal, the absolute *clock_nanosleep*( ) function can be invoked again with the same *rqtp* argument used in the interrupted call.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getres( ), *nanosleep*( ), *pthread_cond_clockwait*( ), *sleep*( )

XBD **<time.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7

The *clock_nanosleep*( ) function is moved from the Clock Selection option to the Base.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0068 [909] is applied.

25616 **NAME**

25617 clock_settime — clock and timer functions

25618 **SYNOPSIS**

25619 CX
```
#include <time.h>

int clock_settime(clockid_t clock_id, const struct timespec *tp);
```

25621 **DESCRIPTION**

25622 Refer to *clock_getres( )*.

25623 **NAME**

25624 clog, clogf, clogl — complex natural logarithm functions

25625 **SYNOPSIS**

25626
```
#include <complex.h>
```

25627
```
double complex clog(double complex z);
```
25628
```
float complex clogf(float complex z);
```
25629
```
long double complex clogl(long double complex z);
```

25630 **DESCRIPTION**

25631 CX The functionality described on this reference page is aligned with the ISO C standard. Any
25632 conflict between the requirements described here and the ISO C standard is unintentional. This
25633 volume of POSIX.1-2024 defers to the ISO C standard.

25634 These functions shall compute the complex natural (base *e*) logarithm of *z*, with a branch cut
25635 along the negative real axis.

25636 **RETURN VALUE**

25637 These functions shall return the complex natural logarithm value, in the range of a strip
25638 mathematically unbounded along the real axis and in the interval [$-i\pi$, $+i\pi$] along the imaginary
25639 axis.

25640 MXC *clog*(*conj*(*z*)), *clogf*(*conjf*(*z*)), and *clogl*(*conjl*(*z*)) shall return exactly the same value as *conj*(*clog*(*z*)),
25641 *conjf*(*clogf*(*z*)), and *conjl*(*clogl*(*z*)), respectively, including for the special values of *z* below.

25642 If *z* is $-0 + i0$, $-\text{Inf} + i\pi$ shall be returned and the divide-by-zero floating-point exception shall be
25643 raised.

25644 If *z* is $+0 + i0$, $-\text{Inf} + i0$ shall be returned and the divide-by-zero floating-point exception shall be
25645 raised.

25646 If *z* is $x + i\text{Inf}$ where *x* is finite, $+\text{Inf} + i\pi/2$ shall be returned.

25647 If *z* is $x + i\text{NaN}$ where *x* is finite, $\text{NaN} + i\text{NaN}$ shall be returned and the invalid floating-point
25648 exception may be raised.

25649 If *z* is $-\text{Inf} + iy$ where *y* is positive-signed and finite, $+\text{Inf} + i\pi$ shall be returned.

25650 If *z* is $+\text{Inf} + iy$ where *y* is positive-signed and finite, $+\text{Inf} + i0$ shall be returned.

25651 If *z* is $-\text{Inf} + i\text{Inf}$, $+\text{Inf} + i3\pi/4$ shall be returned.

25652 If *z* is $+\text{Inf} + i\text{Inf}$, $+\text{Inf} + i\pi/4$ shall be returned.

25653 If *z* is $\pm\text{Inf} + i\text{NaN}$, $+\text{Inf} + i\text{NaN}$ shall be returned.

25654 If *z* is $\text{NaN} + iy$ where *y* is finite, $\text{NaN} + i\text{NaN}$ shall be returned and the invalid floating-point
25655 exception may be raised.

25656 If *z* is $\text{NaN} + i\text{Inf}$, $+\text{Inf} + i\text{NaN}$ shall be returned.

25657 If *z* is $\text{NaN} + i\text{NaN}$, $\text{NaN} + i\text{NaN}$ shall be returned.

25658 **ERRORS**

25659 No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cexp( )

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

25676 **NAME**

25677 close, posix_close — close a file descriptor

25678 **SYNOPSIS**

25679
```
#include <unistd.h>
```

25680
```
int close(int fildes);
```
25681
```
int posix_close(int fildes, int flag);
```

25682 **DESCRIPTION**

25683 The *close*( ) function shall deallocate the file descriptor indicated by *fildes*. To deallocate means to
25684 make the file descriptor available for return by subsequent calls to *open*( ) or other functions that
25685 allocate file descriptors. All process-owned file locks that the calling process owns on the file
25686 associated with the file descriptor shall be unlocked.

25687 If *close*( ) is interrupted by a signal that is to be caught, then it is unspecified whether it returns
25688 –1 with *errno* set to [EINTR] and *fildes* remaining open, or returns –1 with *errno* set to
25689 [EINPROGRESS] and *fildes* being closed, or returns 0 to indicate successful completion; except
25690 that if POSIX_CLOSE_RESTART is defined as 0, then the option of returning –1 with *errno* set to
25691 [EINTR] and *fildes* remaining open shall not occur. If *close*( ) returns –1 with *errno* set to [EINTR],
25692 it is unspecified whether *fildes* can subsequently be passed to any function except *close*( ) or
25693 *posix_close*( ) without error. For all other error situations (except for [EBADF] where *fildes* was
25694 invalid), *fildes* shall be closed. If *fildes* was closed even though the close operation is incomplete,
25695 the close operation shall continue asynchronously and the process shall have no further ability
25696 to track the completion or final status of the close operation.

25697 When all file descriptors associated with a pipe or FIFO special file are closed, any data
25698 remaining in the pipe or FIFO shall be discarded.

25699 When all file descriptors associated with an open file description have been closed, the open file
25700 description shall be freed.

25701 If the link count of the file is 0, when all file descriptors associated with the file are closed, the
25702 space occupied by the file shall be freed and the file shall no longer be accessible.

25703 XSI If *fildes* refers to the manager side of a pseudo-terminal, and this is the last close, a SIGHUP
25704 signal shall be sent to the controlling process, if any, for which the subsidiary side of the pseudo-
25705 terminal is the controlling terminal. It is unspecified whether closing the manager side of the
25706 pseudo-terminal flushes all queued input and output.

25707 When there is an outstanding cancelable asynchronous I/O operation against *fildes* when *close*( )
25708 is called, that I/O operation may be canceled. An I/O operation that is not canceled completes
25709 as if the *close*( ) operation had not yet occurred. All operations that are not canceled shall
25710 complete as if the *close*( ) blocked until the operations completed. The *close*( ) operation itself
25711 need not block awaiting such I/O completion. Whether any I/O operation is canceled, and
25712 which I/O operation may be canceled upon *close*( ), is implementation-defined.

25713 SHM If a memory mapped file or a shared memory object remains referenced at the last close (that is,
25714 a process has it mapped), then the entire contents of the memory object shall persist until the
25715 SHM memory object becomes unreferenced. If this is the last close of a memory mapped file or a
25716 shared memory object and the close results in the memory object becoming unreferenced, and
25717 the memory object has been unlinked, then the memory object shall be removed.

25718 When all file descriptors associated with a socket have been closed, the socket shall be
25719 destroyed. If the socket is in connection-mode, and the SO_LINGER option is set for the socket
25720 with non-zero linger time, and the socket has untransmitted data, then *close*( ) shall block for up
25721 to the current linger interval until all data is transmitted.

The *posix_close*( ) function shall be equivalent to the *close*( ) function, except with the modifications based on the *flag* argument as described below. If *flag* is 0, then *posix_close*( ) shall not return –1 with *errno* set to [EINTR], which implies that *fildes* will always be closed (except for [EBADF], where *fildes* was invalid). If *flag* includes POSIX_CLOSE_RESTART and POSIX_CLOSE_RESTART is defined as a non-zero value, and *posix_close*( ) is interrupted by a signal that is to be caught, then *posix_close*( ) may return –1 with *errno* set to [EINTR], in which case *fildes* shall be left open; however, it is unspecified whether *fildes* can subsequently be passed to any function except *close*( ) or *posix_close*( ) without error. If *flag* is invalid, *posix_close*( ) may fail with errno set to [EINVAL], but shall otherwise behave as if *flag* had been 0 and close *fildes*.

RETURN VALUE

Upon successful completion, 0 shall be returned; otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *close*( ) and *posix_close*( ) functions shall fail if:

[EBADF] The *fildes* argument is not a open file descriptor.

[EINPROGRESS] The function was interrupted by a signal and *fildes* was closed but the close operation is continuing asynchronously.

The *close*( ) and *posix_close*( ) functions may fail if:

[EINTR] The function was interrupted by a signal, POSIX_CLOSE_RESTART is defined as non-zero, and (in the case of *posix_close*( )) the *flag* argument included POSIX_CLOSE_RESTART, in which case *fildes* is still open.

[EIO] An I/O error occurred while reading from or writing to the file system.

The *posix_close*( ) function may fail if:

[EINVAL] The value of the *flag* argument is invalid.

The *close*( ) and *posix_close*( ) functions shall not return an [EAGAIN] or [EWOULDBLOCK] error. If POSIX_CLOSE_RESTART is zero, the *close*( ) function shall not return an [EINTR] error. The *posix_close*( ) function shall not return an [EINTR] error unless *flag* includes a non-zero POSIX_CLOSE_RESTART.

EXAMPLES

Reassigning a File Descriptor

The following example closes the file descriptor associated with standard output for the current process, re-assigns standard output to a new file descriptor, and closes the original file descriptor to clean up. This example assumes that the file descriptor 0 (which is the descriptor for standard input) is not closed.

```
#include <unistd.h>
...
int pfd;
...
close(1);
dup(pfd);
close(pfd);
...
```

Incidentally, this is exactly what could be achieved using:

```
dup2(pfd, 1);
close(pfd);
```

Closing a File Descriptor

In the following example, *close*( ) is used to close a file descriptor after an unsuccessful attempt is made to associate that file descriptor with a stream.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#define LOCKFILE "/etc/ptmp"
...
int pfd;
FILE *fpfd;
...
if ((fpfd = fdopen (pfd, "w")) == NULL) {
    close(pfd);
    unlink(LOCKFILE);
    exit(1);
}
...
```

APPLICATION USAGE

An application that had used the *stdio* routine *fopen*( ) to open a file should use the corresponding *fclose*( ) routine rather than *close*( ). Once a file is closed, the file descriptor no longer exists, since the integer corresponding to it no longer refers to a file.

Implementations may use file descriptors that must be inherited into child processes for the child process to remain conforming, such as for message catalog or tracing purposes. Therefore, an application that calls *close*( ) on an arbitrary integer risks non-conforming behavior, and *close*( ) can only portably be used on file descriptor values that the application has obtained through explicit actions, as well as the three file descriptors corresponding to the standard file streams. In multi-threaded parent applications, the practice of calling *close*( ) in a loop after *fork*( ) and before an *exec* call in order to avoid a race condition of leaking an unintended file descriptor into a child process, is therefore unsafe, and the race should instead be combatted by opening all file descriptors with the FD_CLOEXEC bit set unless the file descriptor is intended to be inherited across *exec*.

Usage of *close*( ) on file descriptors STDIN_FILENO, STDOUT_FILENO, or STDERR_FILENO should immediately be followed by an operation to reopen these file descriptors. Unexpected behavior will result if any of these file descriptors is left in a closed state (for example, an [EBADF] error from *perror*( )) or if an unrelated *open*( ) or similar call later in the application accidentally allocates a file to one of these well-known file descriptors. Furthermore, a *close*( ) followed by a reopen operation (e.g., *open*( ), *dup*( ), etc.) is not atomic; *dup2*( ) should be used to change standard file descriptors.

RATIONALE

The use of interruptible device close routines should be discouraged to avoid problems with the implicit closes of file descriptors, such as by *exec*, process termination, or *dup2*( ). This volume of POSIX.1-2024 only intends to permit such behavior by specifying the [EINTR] error condition for *close*( ) and *posix_close*( ) with non-zero POSIX_CLOSE_RESTART, to allow applications a portable way to resume waiting for an event associated with the close operation (for example, a tape drive rewinding) after receiving an interrupt. This standard also permits implementations

to define POSIX_CLOSE_RESTART to 0 if they do not choose to provide a way to restart an interrupted close action. Although the file descriptor is left open on [EINTR], it might no longer be usable; that is, passing it to any function except *close*( ) or *posix_close*( ) might result in an error such as [EIO]. If an application must guarantee that data will not be lost, it is recommended that the application use *fsync*( ) or *fdatasync*( ) prior to the close operation, rather than leaving the close operation to deal with pending I/O and risk an interrupt.

Earlier versions of this standard left the state of *fildes* unspecified after errors such as [EINTR] and [EIO]; and implementations differed on whether *close*( ) left *fildes* open after [EINTR]. This was unsatisfactory once threads were introduced, since multi-threaded applications need to know whether *fildes* has been closed. Applications cannot blindly call *close*( ) again, because if *fildes* was closed by the first call another thread could have been allocated a file descriptor with the same value as *fildes*, which must not be closed by the first thread. On the other hand, the alternative of never retrying *close*( ) would lead to a file descriptor leak in implementations where *close*( ) did not close *fildes*, although such a leak may be harmless if the process is about to exit or the file descriptor is marked FD_CLOEXEC and the process is about to be replaced by *exec*. This standard introduced *posix_close*( ) with a *flag* argument that allows a choice between the two possible error behaviors, and leaves it unspecified which of the two behaviors is implemented by *close*( ) (although it is guaranteed to be one of the two behaviors of *posix_close*( ), rather than leaving things completely unspecified as in earlier versions of the standard).

Note that the standard requires that *close*( ) and *posix_close*( ) must leave *fildes* open after [EINTR] (in the cases where [EINTR] is permitted) and must close the file descriptor regardless of all other errors (except [EBADF], where *fildes* is already invalid). In general, portable applications should only retry a *close*( ) after checking for [EINTR] (and on implementations where POSIX_CLOSE_RESTART is defined to be zero, this retry loop will be dead code), and risk a file descriptor leak if a retry loop is not attempted. It should also be noted that [EINTR] is only possible if *close*( ) can be interrupted; if no signal handlers are installed, then *close*( ) will not be interrupted. Conversely, if a single-threaded application can guarantee that no file descriptors are opened or closed in signal handlers, then a retry loop without checking for [EINTR] will be harmless (since the retry will fail with [EBADF]), but guaranteeing that no external libraries introduce the use of threading can be difficult. There are additional guarantees for applications which will only ever be used on systems where POSIX_CLOSE_RESTART is defined as 0. These observations should help in determining whether an application needs to have its *close*( ) calls audited for replacement with *posix_close*( ).

It should also be noted that the requirement for *posix_close*( ) with a *flag* of 0 to always close *fildes*, even if an error is reported, is similar to the requirements on *fclose*( ) to always release the stream, even if an error is encountered while flushing data.

Implementations that previously always closed *fildes* can meet the new requirements by translating [EINTR] to [EINPROGRESS] in *close*( ); and may define POSIX_CLOSE_RESTART to 0 rather than having to add restart semantics. On the other hand, implementations that previously left *fildes* open on [EINTR] can map that to *posix_close*( ) with POSIX_CLOSE_RESTART, and must add the semantics of *posix_close*( ) when *flag* is 0; one possibility is by calling the original *close*( ) implementation, checking for failure, and on [EINTR], using actions similar to *dup2*( ) to replace the incomplete close operation with another file descriptor that can be closed immediately by another call to the original *close*( ), all before returning to the application. Either way, *close*( ) should always map to one of the two behaviors of *posix_close*( ), and implementations are encouraged to keep the behavior of *close*( ) unchanged so as not to break implementation-specific expectations of older applications that were relying on behavior not specified by older versions of this standard.

The standard developers considered introducing a thread-local variable that *close*( ) would set to

indicate whether it had closed *fildes* when returning –1. However, this was rejected in favor of the simpler solution of tightening *close*( ) to guarantee that *fildes* is closed except for [EINTR], and exposing a choice of whether to expect [EINTR] by adding *posix_close*( ). Additionally, while the name *posix_close*( ) is new to this standard, it is reminiscent of at least one implementation that introduced an alternate system call named *close_nocancel*( ) in order to allow an application to choose whether restart semantics were desired.

Another consideration was whether implementations might return [EAGAIN] as an extension and whether *close*( ) should be required to leave the file descriptor open in this case, since [EAGAIN] normally implies an operation should be retried. It seemed very unlikely that any implementation would have a legitimate reason to return [EAGAIN] or [EWOULDBLOCK], and therefore this requirement would mean applications have to include code for an error case that will never be used. Therefore *close*( ) is now forbidden from returning [EAGAIN] and [EWOULDBLOCK] errors.

Note that the requirement for *close*( ) on a socket to block for up to the current linger interval is not conditional on the O_NONBLOCK setting.

The standard developers rejected a proposal to add *closefrom*( ) to the standard. Because the standard permits implementations to use inherited file descriptors as a means of providing a conforming environment for the child process, it is not possible to standardize an interface that closes arbitrary file descriptors above a certain value while still guaranteeing a conforming environment.

FUTURE DIRECTIONS

None.

SEE ALSO

dup( ), *exec*, *exit*( ), *fclose*( ), *fopen*( ), *fork*( ), *open*( ), *perror*( ), *unlink*( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

Issue 6

The DESCRIPTION related to a STREAMS-based file or pseudo-terminal is marked as part of the XSI STREAMS Option Group.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] error condition is added as an optional error.
- The DESCRIPTION is updated to describe the state of the *fildes* file descriptor as unspecified if an I/O error occurs and an [EIO] error condition is returned.

Text referring to sockets is added to the DESCRIPTION.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that shared memory objects and memory mapped files (and not typed memory objects) are the types of memory objects to which the paragraph on last closes applies.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/12 is applied, correcting the XSH shaded text relating to the manager side of a pseudo-terminal. The reason for the change is that the behavior of pseudo-terminals and regular terminals should be as much alike as possible in this case; the change achieves that and matches historical behavior.

Issue 7

Functionality relating to the XSI STREAMS option is marked obsolescent.

Functionality relating to the Asynchronous Input and Output and Memory Mapped Files options is moved to the Base.

Austin Group Interpretation 1003.1-2001 #139 is applied, clarifying that the requirement for *close*( ) on a socket to block for up to the current linger interval is not conditional on the O_NONBLOCK setting.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0059 [419], XSH/TC1-2008/0060 [149], and XSH/TC1-2008/0061 [149] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0069 [555] is applied.

Issue 8

Austin Group Defect 529 is applied, adding the *posix_close*( ) function and changing requirements for the *close*( ) function relating to [EINTR].

Austin Group Defect 768 is applied, adding OFD-owned file locks.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1466 is applied, changing the terminology used for pseudo-terminal devices.

Austin Group Defect 1525 is applied, clarifying that a socket is not destroyed until all file descriptors associated with it have been closed.

NAME

closedir — close a directory stream

SYNOPSIS

```
#include <dirent.h>

int closedir(DIR *dirp);
```

DESCRIPTION

The *closedir*( ) function shall close the directory stream referred to by the argument *dirp*. Upon return, the value of *dirp* may no longer point to an accessible object of the type **DIR**. If there is a file descriptor associated with the stream (whether opened by *opendir*( ) or *dirfd*( ), or passed to *fdopendir*( ) when creating the stream), that file descriptor shall be closed by *closedir*( ).

RETURN VALUE

Upon successful completion, *closedir*( ) shall return 0; otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *closedir*( ) function may fail if:

[EBADF] The *dirp* argument does not refer to an open directory stream.

[EINTR] The *closedir*( ) function was interrupted by a signal.

EXAMPLES

Closing a Directory Stream

The following program fragment demonstrates how the *closedir*( ) function is used.

```
...
    DIR *dir;
    struct dirent *dp;
...
    if ((dir = opendir (".")) == NULL) {
...
    }

    while ((dp = readdir (dir)) != NULL) {
...
    }

    closedir(dir);
...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

dirfd( ), *fdopendir*( )

XBD **<dirent.h>**

25966 **CHANGE HISTORY**

25967 First released in Issue 2.

25968 **Issue 6**

25969 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

25970 The following new requirements on POSIX implementations derive from alignment with the
25971 Single UNIX Specification:

25972 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
25973 required for conforming implementations of previous POSIX specifications, it was not
25974 required for UNIX applications.

25975 • The [EINTR] error condition is added as an optional error condition.

25976 **Issue 8**

25977 Austin Group Defect 1360 is applied, clarifying that type **DIR** always has the ability to store a
25978 file descriptor; what is optional is whether one is opened by *opendir*( ).

NAME

closelog, openlog, setlogmask, syslog — control system log

SYNOPSIS

XSI
```
#include <syslog.h>

void closelog(void);
void openlog(const char *ident, int logopt, int facility);
int setlogmask(int maskpri);
void syslog(int priority, const char *message, ... /* arguments */);
```

DESCRIPTION

The *syslog*( ) function shall send a message to an implementation-defined logging facility, which may log it in an implementation-defined system log, write it to the system console, forward it to a list of users, or forward it to the logging facility on another host over the network. The logged message shall include a message header and a message body. The message header contains at least a timestamp and a tag string.

The message body is generated from the *message* and following arguments in the same manner as if these were arguments to *printf*( ), except that the additional conversion specification `%m` shall be recognized; it shall convert no arguments, shall cause the output of the error message string associated with the value of *errno* on entry to *syslog*( ), and may be mixed with argument specifications of the "`%`*n*`$`" form. If a complete conversion specification with the `m` conversion specifier character is not just `%m`, the behavior is undefined. A trailing <newline> may be added if needed.

Values of the *priority* argument are formed by OR'ing together a severity-level value and an optional facility value. If no facility value is specified, the current default facility value is used.

Possible values of severity level include:

LOG_EMERG A panic condition.

LOG_ALERT A condition that should be corrected immediately, such as a corrupted system database.

LOG_CRIT Critical conditions, such as hard device errors.

LOG_ERR Errors.

LOG_WARNING
Warning messages.

LOG_NOTICE Conditions that are not error conditions, but that may require special handling.

LOG_INFO Informational messages.

LOG_DEBUG Messages that contain information normally of use only when debugging a program.

The facility indicates the application or system component generating the message. Possible facility values include:

LOG_USER Messages generated by arbitrary processes. This is the default facility identifier if none is specified.

LOG_LOCAL0 Reserved for local use.

LOG_LOCAL1 Reserved for local use.

LOG_LOCAL2 Reserved for local use.

LOG_LOCAL3 Reserved for local use.

LOG_LOCAL4 Reserved for local use.

LOG_LOCAL5 Reserved for local use.

LOG_LOCAL6 Reserved for local use.

LOG_LOCAL7 Reserved for local use.

The *openlog*( ) function shall set process attributes that affect subsequent calls to *syslog*( ). The *ident* argument is a a pointer to a null-terminated identifier that shall be prepended (without the null terminator) to every message. The application shall ensure that the string pointed to by *ident* remains valid during the *syslog*( ) calls that will prepend this identifier; however, it is unspecified whether changes made to the string will change the identifier prepended by later *syslog*( ) calls. The *logopt* argument indicates logging options. Values for *logopt* are constructed by a bitwise-inclusive OR of zero or more of the following:

LOG_PID Log the process ID with each message. This is useful for identifying specific processes.

LOG_CONS Write messages to the system console if they cannot be sent to the logging facility. The *syslog*( ) function ensures that the process does not acquire the console as a controlling terminal in the process of writing the message.

LOG_NDELAY Open the connection to the logging facility immediately. Normally the open is delayed until the first message is logged. This is useful for programs that need to manage the order in which file descriptors are allocated.

LOG_ODELAY Delay open until *syslog*( ) is called.

LOG_NOWAIT Do not wait for child processes that may have been created during the course of logging the message. This option should be used by processes that enable notification of child termination using SIGCHLD, since *syslog*( ) may otherwise block waiting for a child whose exit status has already been collected.

The *facility* argument encodes a default facility to be assigned to all messages that do not have an explicit facility already encoded. The initial default facility is LOG_USER.

The *openlog*( ) and *syslog*( ) functions may allocate a file descriptor. It is not necessary to call *openlog*( ) prior to calling *syslog*( ). If a file descriptor is allocated, the FD_CLOEXEC flag shall be set; see **<fcntl.h>**.

The *closelog*( ) function shall close any open file descriptors allocated by previous calls to *openlog*( ) or *syslog*( ).

The *setlogmask*( ) function shall set the log priority mask for the current process to *maskpri* and return the previous mask. If the *maskpri* argument is 0, the current log mask is not modified. Calls by the current process to *syslog*( ) with a priority not set in *maskpri* shall be rejected. The default log mask allows all priorities to be logged. A call to *openlog*( ) is not required prior to calling *setlogmask*( ).

The LOG_MASK(*pri*) and LOG_UPTO(*pri*) macros can be used to ensure a value or range of severity levels is properly encoded for the *setlogmask*( ) *maskpri* argument in a portable manner. The masks produced by these macros can be OR'ed or AND'ed with other priority masks (for

example,

```
LOG_UPTO(LOG_WARNING) | LOG_MASK(LOG_DEBUG)
```

and

```
LOG_UPTO(LOG_DEBUG) & ~((LOG_MASK(LOG_NOTICE) | LOG_MASK(LOG_INFO))
```

would produce the same priority mask).

Symbolic constants for use as values of the *logopt*, *facility*, *priority*, and *maskpri* arguments are defined in the **<syslog.h>** header.

RETURN VALUE

The *setlogmask*( ) function shall return the previous log priority mask. The *closelog*( ), *openlog*( ), and *syslog*( ) functions shall not return a value.

ERRORS

The *openlog*( ) and *syslog*( ) functions may fail if:

[EMFILE] All file descriptors available to the process are currently open.

[ENFILE] The maximum allowable number of files is currently open in the system.

EXAMPLES

Using openlog( )

The following example causes subsequent calls to *syslog*( ) to log the process ID with each message, and to write messages to the system console if they cannot be sent to the logging facility.

```
#include <syslog.h>

char *ident = "Process demo";
int logopt = LOG_PID | LOG_CONS;
int facility = LOG_USER;
...
openlog(ident, logopt, facility);
```

Using setlogmask( )

The following example causes subsequent calls to *syslog*( ) to accept error messages, and to reject all other messages.

```
#include <syslog.h>

int result;
int mask = LOG_MASK (LOG_ERR);
...
result = setlogmask(mask);
```


Using syslog

The following example sends the message `"This is a message"` to the default logging facility, marking the message as an error message generated by random processes.

```
#include <syslog.h>

char *message = "This is a message";
int priority = LOG_ERR | LOG_USER;
...
syslog(priority, message);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fprintf( )

XBD **<fcntl.h>**, **<syslog.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/13 is applied, correcting the EXAMPLES section.

Issue 8

Austin Group Defect 368 is applied, adding a requirement for FD_CLOEXEC to be set if a file descriptor is allocated, and adding the [EMFILE] and [ENFILE] errors.

Austin Group Defect 1033 is applied, adding the LOG_UPTO macro.

Austin Group Defect 1244 is applied, clarifying the handling of the *ident* argument.

NAME

cnd_broadcast, cnd_signal — broadcast or signal a condition

SYNOPSIS

```
#include <threads.h>

int cnd_broadcast(cnd_t *cond);
int cnd_signal(cnd_t *cond);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *cnd_broadcast*( ) function shall, as a single atomic operation, determine which threads, if any, are blocked on the condition variable pointed to by *cond* and unblock all of these threads.

CX The *cnd_signal*( ) function shall, as a single atomic operation, determine which threads, if any, are blocked on the condition variable pointed to by *cond* and unblock at least one of these threads.

If these functions determine that there are no threads blocked on the condition variable pointed to by *cond*, they shall have no effect and shall return `thrd_success`.

CX If more than one thread is blocked on a condition variable, the scheduling policy shall determine the order in which threads are unblocked. When each thread unblocked as a result of a *cnd_broadcast*( ) or *cnd_signal*( ) returns from its call to *cnd_wait*( ) or *cnd_timedwait*( ), the thread shall own the mutex with which it called *cnd_wait*( ) or *cnd_timedwait*( ). The thread(s) that are unblocked shall contend for the mutex according to the scheduling policy (if applicable), and as if each had called *mtx_lock*( ).

The *cnd_broadcast*( ) and *cnd_signal*( ) functions can be called by a thread whether or not it currently owns the mutex that threads calling *cnd_wait*( ) or *cnd_timedwait*( ) have associated with the condition variable during their waits; however, if predictable scheduling behavior is required, then that mutex shall be locked by the thread calling *cnd_broadcast*( ) or *cnd_signal*( ).

These functions shall not be affected if the calling thread executes a signal handler during the call.

The behavior is undefined if the value specified by the *cond* argument to *cnd_broadcast*( ) or *cnd_signal*( ) does not refer to an initialized condition variable.

RETURN VALUE

These functions shall return `thrd_success` on success, or `thrd_error` if the request could not be honored.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

See the APPLICATION USAGE section for *pthread_cond_broadcast*( ), substituting *cnd_broadcast*( ) for *pthread_cond_broadcast*( ) and *cnd_signal*( ) for *pthread_cond_signal*( ).

26166 **RATIONALE**

26167 As for *pthread_cond_broadcast*( ) and *pthread_cond_signal*( ), spurious wakeups may occur with
26168 *cnd_broadcast*( ) and *cnd_signal*( ), necessitating that applications code a predicate-testing-loop
26169 around the condition wait. (See the RATIONALE section for *pthread_cond_broadcast*( ).)

26170 These functions are not affected by signal handlers for the reasons stated in XRAT Section B.2.3
26171 (on page 3742).

26172 **FUTURE DIRECTIONS**

26173 None.

26174 **SEE ALSO**

26175 *cnd_destroy*( ), *cnd_timedwait*( ), *pthread_cond_broadcast*( )

26176 XBD Section 4.15.2, **<threads.h>**

26177 **CHANGE HISTORY**

26178 First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

cnd_destroy, cnd_init — destroy and initialize condition variables

SYNOPSIS

```
#include <threads.h>

void cnd_destroy(cnd_t *cond);
int  cnd_init(cnd_t *cond);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *cnd_destroy*( ) function shall release all resources used by the condition variable pointed to by *cond*. It shall be safe to destroy an initialized condition variable upon which no threads are currently blocked. Attempting to destroy a condition variable upon which other threads are currently blocked results in undefined behavior. A destroyed condition variable object can be reinitialized using *cnd_init*( ); the results of otherwise referencing the object after it has been destroyed are undefined. The behavior is undefined if the value specified by the *cond* argument to *cnd_destroy*( ) does not refer to an initialized condition variable.

The *cnd_init*( ) function shall initialize a condition variable. If it succeeds it shall set the variable pointed to by *cond* to a value that uniquely identifies the newly initialized condition variable. Attempting to initialize an already initialized condition variable results in undefined behavior. A thread that calls *cnd_wait*( ) on a newly initialized condition variable shall block.

CX See Section 2.9.9 (on page 548) for further requirements.

These functions shall not be affected if the calling thread executes a signal handler during the call.

RETURN VALUE

The *cnd_destroy*( ) function shall not return a value.

The *cnd_init*( ) function shall return `thrd_success` on success, or `thrd_nomem` if no memory could be allocated for the newly created condition variable, or `thrd_error` if the request could not be honored.

ERRORS

See RETURN VALUE.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

These functions are not affected by signal handlers for the reasons stated in XRAT Section B.2.3 (on page 3742).

FUTURE DIRECTIONS

None.

SEE ALSO

cnd_broadcast( ), *cnd_timedwait*( )

XBD **<threads.h>**

26222 **CHANGE HISTORY**

26223 First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

cnd_timedwait, cnd_wait — wait on a condition

SYNOPSIS

```
#include <threads.h>

int cnd_timedwait(cnd_t * restrict cond, mtx_t * restrict mtx,
    const struct timespec * restrict ts);
int cnd_wait(cnd_t *cond, mtx_t *mtx);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *cnd_timedwait*( ) function shall atomically unlock the mutex pointed to by *mtx* and block until the condition variable pointed to by *cond* is signaled by a call to *cnd_signal*( ) or to *cnd_broadcast*( ), or until after the TIME_UTC-based calendar time pointed to by *ts*, or until it is unblocked due to an unspecified reason.

The *cnd_wait*( ) function shall atomically unlock the mutex pointed to by *mtx* and block until the condition variable pointed to by *cond* is signaled by a call to *cnd_signal*( ) or to *cnd_broadcast*( ), or until it is unblocked due to an unspecified reason.

CX Atomically here means ``atomically with respect to access by another thread to the mutex and then the condition variable''. That is, if another thread is able to acquire the mutex after the about-to-block thread has released it, then a subsequent call to *cnd_broadcast*( ) or *cnd_signal*( ) in that thread shall behave as if it were issued after the about-to-block thread has blocked.

When the calling thread becomes unblocked, these functions shall lock the mutex pointed to by *mtx* before they return. The application shall ensure that the mutex pointed to by *mtx* is locked by the calling thread before it calls these functions.

When using condition variables there is always a Boolean predicate involving shared variables associated with each condition wait that is true if the thread should proceed. Spurious wakeups from the *cnd_timedwait*( ) and *cnd_wait*( ) functions may occur. Since the return from *cnd_timedwait*( ) or *cnd_wait*( ) does not imply anything about the value of this predicate, the predicate should be re-evaluated upon such return.

When a thread waits on a condition variable, having specified a particular mutex to either the *cnd_timedwait*( ) or the *cnd_wait*( ) operation, a dynamic binding is formed between that mutex and condition variable that remains in effect as long as at least one thread is blocked on the condition variable. During this time, the effect of an attempt by any thread to wait on that condition variable using a different mutex is undefined. Once all waiting threads have been unblocked (as by the *cnd_broadcast*( ) operation), the next wait operation on that condition variable shall form a new dynamic binding with the mutex specified by that wait operation. Even though the dynamic binding between condition variable and mutex might be removed or replaced between the time a thread is unblocked from a wait on the condition variable and the time that it returns to the caller or begins cancellation cleanup, the unblocked thread shall always re-acquire the mutex specified in the condition wait operation call from which it is returning.

CX A condition wait (whether timed or not) is a cancellation point. When the cancelability type of a thread is set to PTHREAD_CANCEL_DEFERRED, a side-effect of acting upon a cancellation request while in a condition wait is that the mutex is (in effect) re-acquired before calling the first cancellation cleanup handler. The effect is as if the thread were unblocked, allowed to execute up to the point of returning from the call to *cnd_timedwait*( ) or *cnd_wait*( ), but at that point notices

the cancellation request and instead of returning to the caller of *cnd_timedwait*( ) or *cnd_wait*( ), starts the thread cancellation activities, which includes calling cancellation cleanup handlers.

A thread that has been unblocked because it has been canceled while blocked in a call to *cnd_timedwait*( ) or *cnd_wait*( ) shall not consume any condition signal that may be directed concurrently at the condition variable if there are other threads blocked on the condition variable.

When *cnd_timedwait*( ) times out, it shall nonetheless release and re-acquire the mutex referenced by *mtx*, and may consume a condition signal directed concurrently at the condition variable.

CX These functions shall not be affected if the calling thread executes a signal handler during the call, except that if a signal is delivered to a thread waiting for a condition variable, upon return from the signal handler either the thread shall resume waiting for the condition variable as if it was not interrupted, or it shall return `thrd_success` due to spurious wakeup.

The behavior is undefined if the value specified by the *cond* or *mtx* argument to these functions does not refer to an initialized condition variable or an initialized mutex object, respectively.

RETURN VALUE

The *cnd_timedwait*( ) function shall return `thrd_success` upon success, or `thrd_timedout` if the time specified in the call was reached without acquiring the requested resource, or `thrd_error` if the request could not be honored.

The *cnd_wait*( ) function shall return `thrd_success` upon success or `thrd_error` if the request could not be honored.

ERRORS

See RETURN VALUE.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

These functions are not affected by signal handlers (except as stated in the DESCRIPTION) for the reasons stated in XRAT Section B.2.3 (on page 3742).

FUTURE DIRECTIONS

None.

SEE ALSO

cnd_broadcast( ), *cnd_destroy*( ), *timespec_get*( )

XBD Section 4.15.2, **<threads.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

confstr — get configurable variables

SYNOPSIS

```
#include <unistd.h>

size_t confstr(int name, char *buf, size_t len);
```

DESCRIPTION

The *confstr*( ) function shall return configuration-defined string values. Its use and purpose are similar to *sysconf*( ), but it is used where string values rather than numeric values are returned.

The *name* argument represents the system variable to be queried. The implementation shall support the following name values, defined in **<unistd.h>**. It may support others:

_CS_PATH
_CS_POSIX_V8_ILP32_OFF32_CFLAGS
_CS_POSIX_V8_ILP32_OFF32_LDFLAGS
_CS_POSIX_V8_ILP32_OFF32_LIBS
_CS_POSIX_V8_ILP32_OFFBIG_CFLAGS
_CS_POSIX_V8_ILP32_OFFBIG_LDFLAGS
_CS_POSIX_V8_ILP32_OFFBIG_LIBS
_CS_POSIX_V8_LP64_OFF64_CFLAGS
_CS_POSIX_V8_LP64_OFF64_LDFLAGS
_CS_POSIX_V8_LP64_OFF64_LIBS
_CS_POSIX_V8_LPBIG_OFFBIG_CFLAGS
_CS_POSIX_V8_LPBIG_OFFBIG_LDFLAGS
_CS_POSIX_V8_LPBIG_OFFBIG_LIBS
_CS_POSIX_V8_THREADS_CFLAGS
_CS_POSIX_V8_THREADS_LDFLAGS
_CS_POSIX_V8_WIDTH_RESTRICTED_ENVS
_CS_V8_ENV

OB
_CS_POSIX_V7_ILP32_OFF32_CFLAGS
_CS_POSIX_V7_ILP32_OFF32_LDFLAGS
_CS_POSIX_V7_ILP32_OFF32_LIBS
_CS_POSIX_V7_ILP32_OFFBIG_CFLAGS
_CS_POSIX_V7_ILP32_OFFBIG_LDFLAGS
_CS_POSIX_V7_ILP32_OFFBIG_LIBS
_CS_POSIX_V7_LP64_OFF64_CFLAGS
_CS_POSIX_V7_LP64_OFF64_LDFLAGS
_CS_POSIX_V7_LP64_OFF64_LIBS
_CS_POSIX_V7_LPBIG_OFFBIG_CFLAGS
_CS_POSIX_V7_LPBIG_OFFBIG_LDFLAGS
_CS_POSIX_V7_LPBIG_OFFBIG_LIBS
_CS_POSIX_V7_THREADS_CFLAGS
_CS_POSIX_V7_THREADS_LDFLAGS
_CS_POSIX_V7_WIDTH_RESTRICTED_ENVS
_CS_V7_ENV

If *len* is not 0, and if *name* has a configuration-defined value, *confstr*( ) shall copy that value into the *len*-byte buffer pointed to by *buf*. If the string to be returned is longer than *len* bytes, including the terminating null, then *confstr*( ) shall truncate the string to *len*–1 bytes and null-terminate the result. The application can detect that the string was truncated by comparing the value returned by *confstr*( ) with *len*.

If *len* is 0 and *buf* is a null pointer, then *confstr*( ) shall still return the integer value as defined below, but shall not return a string. If *len* is 0 but *buf* is not a null pointer, the result is unspecified.

After a call to:

```
confstr(_CS_V8_ENV, buf, sizeof(buf))
```

the string stored in *buf* shall contain a <space>-separated list of the variable=value environment variable pairs an implementation requires as part of specifying a conforming environment, as described in the implementations' conformance documentation.

If the implementation supports the POSIX shell option, the string stored in *buf* after a call to:

```
confstr(_CS_PATH, buf, sizeof(buf))
```

can be used as a value of the *PATH* environment variable that accesses all of the standard utilities of POSIX.1-2024, that are provided in a manner accessible via the *exec* family of functions, if the return value is less than or equal to *sizeof*(*buf*).

RETURN VALUE

If *name* has a configuration-defined value, *confstr*( ) shall return the size of buffer that would be needed to hold the entire configuration-defined value including the terminating null. If this return value is greater than *len*, the string returned in *buf* is truncated.

If *name* is invalid, *confstr*( ) shall return 0 and set *errno* to indicate the error.

If *name* does not have a configuration-defined value, *confstr*( ) shall return 0 and leave *errno* unchanged.

ERRORS

The *confstr*( ) function shall fail if:

[EINVAL] The value of the *name* argument is invalid.

EXAMPLES

None.

APPLICATION USAGE

An application can distinguish between an invalid *name* parameter value and one that corresponds to a configurable variable that has no configuration-defined value by checking if *errno* is modified. This mirrors the behavior of *sysconf*( ).

The original need for this function was to provide a way of finding the configuration-defined default value for the environment variable *PATH*. Since *PATH* can be modified by the user to include directories that could contain utilities replacing the standard utilities in the Shell and Utilities volume of POSIX.1-2024, applications need a way to determine the system-supplied *PATH* environment variable value that contains the correct search path for the standard utilities.

An application could use:

```
confstr(name, (char *)NULL, (size_t)0)
```

to find out how big a buffer is needed for the string value; use *malloc*( ) to allocate a buffer to hold the string; and call *confstr*( ) again to get the string. Alternately, it could allocate a fixed, static buffer that is big enough to hold most answers (perhaps 512 or 1 024 bytes), but then use *malloc*( ) to allocate a larger buffer if it finds that this is too small.

RATIONALE

Application developers can normally determine any configuration variable by means of reading from the stream opened by a call to:

```
popen("command -p getconf variable", "r");
```

The *confstr*( ) function with a *name* argument of _CS_PATH returns a string that can be used as a *PATH* environment variable setting that will reference the standard shell and utilities as described in the Shell and Utilities volume of POSIX.1-2024.

The *confstr*( ) function copies the returned string into a buffer supplied by the application instead of returning a pointer to a string. This allows a cleaner function in some implementations (such as those with lightweight threads) and resolves questions about when the application must copy the string returned.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fpathconf*( ), *sysconf*( )

XBD **<unistd.h>**

XCU *c17*

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 5

A table indicating the permissible values of *name* is added to the DESCRIPTION. All those marked EX are new in this version.

Issue 6

The Open Group Corrigendum U033/7 is applied. The return value for the case returning the size of the buffer now explicitly states that this includes the terminating null.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated with new arguments which can be used to determine configuration strings for C compiler flags, linker/loader flags, and libraries for each different supported programming environment. This is a change to support data size neutrality.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The DESCRIPTION is updated to include text describing how _CS_PATH can be used to obtain a *PATH* to access the standard utilities.

The macros associated with the *c89* programming models are marked LEGACY and new equivalent macros associated with *c99* are introduced.

Issue 7

Austin Group Interpretation 1003.1-2001 #047 is applied, adding the _CS_V7_ENV variable.

Austin Group Interpretations 1003.1-2001 #166 is applied to permit an additional compiler flag to enable threads.

The V6 variables for the supported programming environments are marked obsolescent.

The variables for the supported programming environments are updated to be V7.

The LEGACY variables and obsolescent values are removed.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0070 [810] and XSH/TC2-2008/0071 [911] are applied.

Issue 8

Austin Group Defect 1330 is applied, changing ``_V7_'' to ``_V8_'' and ``_V6_'' to ``_V7_''.

NAME

conj, conjf, conjl — complex conjugate functions

SYNOPSIS

```
#include <complex.h>

double complex conj(double complex z);
float complex conjf(float complex z);
long double complex conjl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the complex conjugate of *z*, by reversing the sign of its imaginary part.

RETURN VALUE

These functions return the complex conjugate value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

carg( ), *cimag*( ), *cproj*( ), *creal*( )

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

NAME

connect — connect a socket

SYNOPSIS

```
#include <sys/socket.h>

int connect(int socket, const struct sockaddr *address,
    socklen_t address_len);
```

DESCRIPTION

The *connect*( ) function shall attempt to make a connection on a connection-mode socket or to set or reset the peer address of a connectionless-mode socket. The function takes the following arguments:

socket Specifies the file descriptor associated with the socket.

address Points to a **sockaddr** structure containing the peer address. The length and format of the address depend on the address family of the socket.

address_len Specifies the length of the **sockaddr** structure pointed to by the *address* argument.

If the address family of the socket is AF_UNIX, the application shall ensure that a null terminator after the pathname is included in the *sun_path* member of *address* as a **sockaddr_un** structure, and that *address_len* is at least `offsetof(struct sockaddr_un, sun_path) + 1` plus the length of the pathname.

If the socket has not already been bound to a local address, *connect*( ) shall bind it to an address which, unless the socket's address family is AF_UNIX, is an unused local address.

If the initiating socket is not connection-mode, then *connect*( ) shall set the socket's peer address, and no connection is made. For SOCK_DGRAM sockets, the peer address identifies where all datagrams are sent on subsequent *send*( ) functions, and limits the remote sender for subsequent *recv*( ) functions. If the *sa_family* member of *address* is AF_UNSPEC, the socket's peer address shall be reset. Note that despite no connection being made, the term ``connected'' is used to describe a connectionless-mode socket for which a peer address has been set.

If the initiating socket is connection-mode, then *connect*( ) shall attempt to establish a connection to the address specified by the *address* argument. If the connection cannot be established immediately and O_NONBLOCK is not set for the file descriptor for the socket, *connect*( ) shall block for up to an unspecified timeout interval until the connection is established. If the timeout interval expires before the connection is established, *connect*( ) shall fail and the connection attempt shall be aborted. If *connect*( ) is interrupted by a signal that is caught while blocked waiting to establish a connection, *connect*( ) shall fail and set *errno* to [EINTR], but the connection request shall not be aborted, and the connection shall be established asynchronously.

If the connection cannot be established immediately and O_NONBLOCK is set for the file descriptor for the socket, *connect*( ) shall fail and set *errno* to [EINPROGRESS], but the connection request shall not be aborted, and the connection shall be established asynchronously. Subsequent calls to *connect*( ) for the same socket, before the connection is established, shall fail and set *errno* to [EALREADY].

When the connection has been established asynchronously, *pselect*( ), *select*( ), *poll*( ), and *ppoll*( ) shall indicate that the file descriptor for the socket is ready for writing.

The socket in use may require the process to have appropriate privileges to use the *connect*( ) function.

26516 **RETURN VALUE**

26517 Upon successful completion, *connect*( ) shall return 0; otherwise, –1 shall be returned and *errno*
26518 set to indicate the error.

26519 **ERRORS**

26520 The *connect*( ) function shall fail if:

26521 [EADDRNOTAVAIL]
26522 The specified address is not available from the local machine.

26523 [EAFNOSUPPORT]
26524 The specified address is not a valid address for the address family of the
26525 specified socket.

26526 [EALREADY] A connection request is already in progress for the specified socket.

26527 [EBADF] The *socket* argument is not a valid file descriptor.

26528 [ECONNREFUSED]
26529 The target address was not listening for connections or refused the connection
26530 request.

26531 [EINPROGRESS] O_NONBLOCK is set for the file descriptor for the socket and the connection
26532 cannot be immediately established; the connection shall be established
26533 asynchronously.

26534 [EINTR] The attempt to establish a connection was interrupted by delivery of a signal
26535 that was caught; the connection shall be established asynchronously.

26536 [EISCONN] The specified socket is connection-mode and is already connected.

26537 [ENETUNREACH]
26538 No route to the network is present.

26539 [ENOTSOCK] The *socket* argument does not refer to a socket.

26540 [EPROTOTYPE] The specified address has a different type than the socket bound to the
26541 specified peer address.

26542 [ETIMEDOUT] The attempt to connect timed out before a connection was made.

26543 If the address family of the socket is AF_UNIX, then *connect*( ) shall fail if:

26544 [EIO] An I/O error occurred while reading from or writing to the file system.

26545 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname
26546 in *address*.

26547 [ENAMETOOLONG]
26548 The length of a component of a pathname is longer than {NAME_MAX}.

26549 [ENOENT] A component of the pathname does not name an existing file or the pathname
26550 is an empty string.

26551 [ENOTDIR] A component of the path prefix of the pathname in *address* names an existing
26552 file that is neither a directory nor a symbolic link to a directory, or the
26553 pathname in *address* contains at least one non-<slash> character and ends with
26554 one or more trailing <slash> characters and the last pathname component
26555 names an existing file that is neither a directory nor a symbolic link to a
26556 directory.

26557 The *connect*( ) function may fail if:

[EACCES] Search permission is denied for a component of the path prefix; or write access to the named socket is denied.

[EADDRINUSE] Attempt to establish a connection that uses addresses that are already in use.

[ECONNRESET] Remote host reset the connection request.

[EHOSTUNREACH]
The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).

[EINVAL] The *address_len* argument is not a valid length for the address family; or invalid address family in the **sockaddr** structure.

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in *address*.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

[ENETDOWN] The local network interface used to reach the destination is down.

[ENOBUFS] No buffer space is available.

[EOPNOTSUPP] The socket is listening and cannot be connected.

EXAMPLES

None.

APPLICATION USAGE

If *connect*( ) fails, the state of the socket is unspecified. Conforming applications should close the file descriptor and create a new socket before attempting to reconnect.

For AF_UNIX sockets, some implementations support an extension where *address_len* does not have to include a null terminator for the pathname stored in *sun_path*, which in turn allows a pathname to be one byte longer. However, such usage is not portable, and carries a risk of accessing beyond the intended bounds of the pathname length.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

accept( ), *bind*( ), *close*( ), *getsockname*( ), *poll*( ), *pselect*( ), *send*( ), *shutdown*( ), *socket*( )

XBD **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #035 is applied, clarifying the description of connected sockets.

Austin Group Interpretation 1003.1-2001 #143 is applied.

Austin Group Interpretation 1003.1-2001 #188 is applied, changing the method used to reset a peer address for a datagram socket.

SD5-XSH-ERN-185 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0062 [324] is applied.

Issue 8

Austin Group Defect 561 is applied, changing the requirements for the *sun_path* member of the **sockaddr_un** structure.

Austin Group Defect 1263 is applied, adding *ppoll*( ).

NAME

copysign, copysignf, copysignl — number manipulation function

SYNOPSIS

```
#include <math.h>

double copysign(double x, double y);
float copysignf(float x, float y);
long double copysignl(long double x, long double y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall produce a value with the magnitude of x and the sign of y. On implementations that represent a signed zero but do not treat negative zero consistently in arithmetic operations, these functions regard the sign of zero as positive.

RETURN VALUE

Upon successful completion, these functions shall return a value with the magnitude of x and the sign of y.

MX The returned value shall be exact and shall be independent of the current rounding direction mode.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

signbit( )

XBD **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

NAME

cos, cosf, cosl — cosine function

SYNOPSIS

```
#include <math.h>

double cos(double x);
float cosf(float x);
long double cosl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the cosine of their argument x, measured in radians.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the cosine of x.

MX If x is NaN, a NaN shall be returned.

If x is ±0, the value 1.0 shall be returned.

If x is ±Inf, a domain error shall occur, and a NaN shall be returned.

ERRORS

These functions shall fail if:

MX Domain Error — The x argument is ±Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

EXAMPLES

Taking the Cosine of a 45-Degree Angle

```
#include <math.h>
...
double radians = 45 * M_PI / 180;
double result;
...
result = cos(radians);
```

APPLICATION USAGE

These functions may lose accuracy when their argument is near an odd multiple of $\pi/2$ or is far from 0.

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

acos( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *sin*( ), *tan*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *cosf*( ) and *cosl*( ) functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0063 [320] is applied.

NAME

cosh, coshf, coshl — hyperbolic cosine functions

SYNOPSIS

```
#include <math.h>

double cosh(double x);
float coshf(float x);
long double coshl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the hyperbolic cosine of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the hyperbolic cosine of *x*.

If the correct value would cause overflow, a range error shall occur and *cosh*( ), *coshf*( ), and *coshl*( ) shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, the value 1.0 shall be returned.

If *x* is ±Inf, +Inf shall be returned.

ERRORS

These functions shall fail if:

Range Error — The result would cause an overflow.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

acosh( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *sinh*( ), *tanh*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *coshf*( ) and *coshl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0072 [630] is applied.

26762 **NAME**

26763 cosl — cosine function

26764 **SYNOPSIS**

26765 `#include <math.h>`

26766 `long double cosl(long double x);`

26767 **DESCRIPTION**

26768 Refer to *cos( )*.

NAME

cpow, cpowf, cpowl — complex power functions

SYNOPSIS

```
#include <complex.h>

double complex cpow(double complex x, double complex y);
float complex cpowf(float complex x, float complex y);
long double complex cpowl(long double complex x,
    long double complex y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the complex power function x^y, with a branch cut for the first parameter along the negative real axis.

MXC These functions shall raise floating-point exceptions if appropriate for the calculation of the parts of the result, and may also raise spurious floating-point exceptions.

RETURN VALUE

These functions shall return the complex power function value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Permitting spurious floating-point exceptions allows *cpow*(*z*, *c*) to be implemented as *cexp*(*c* *clog*(*z*)) without precluding implementations that treat special cases more carefully.

FUTURE DIRECTIONS

None.

SEE ALSO

cabs( ), *csqrt*( )

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

NAME

cproj, cprojf, cprojl — complex projection functions

SYNOPSIS

```
#include <complex.h>

double complex cproj(double complex z);
float complex cprojf(float complex z);
long double complex cprojl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute a projection of *z* onto the Riemann sphere: *z* projects to *z*, except that all complex infinities (even those with one infinite part and one NaN part) project to positive infinity on the real axis. If *z* has an infinite part, then *cproj*(*z*) shall be equivalent to:

```
INFINITY + I * copysign(0.0, cimag(z))
```

RETURN VALUE

These functions shall return the value of the projection onto the Riemann sphere.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Two topologies are commonly used in complex mathematics: the complex plane with its continuum of infinities, and the Riemann sphere with its single infinity. The complex plane is better suited for transcendental functions, the Riemann sphere for algebraic functions. The complex types with their multiplicity of infinities provide a useful (though imperfect) model for the complex plane. The *cproj*( ) function helps model the Riemann sphere by mapping all infinities to one, and should be used just before any operation, especially comparisons, that might give spurious results for any of the other infinities. Note that a complex value with one infinite part and one NaN part is regarded as an infinity, not a NaN, because if one part is infinite, the complex value is infinite independent of the value of the other part. For the same reason, *cabs*( ) returns an infinity if its argument has an infinite part and a NaN part.

FUTURE DIRECTIONS

None.

SEE ALSO

carg( ), *cimag*( ), *conj*( ), *creal*( )

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

NAME

creal, crealf, creall — complex real functions

SYNOPSIS

```
#include <complex.h>

double creal(double complex z);
float crealf(float complex z);
long double creall(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the real part of *z*.

RETURN VALUE

These functions shall return the real part value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

For a variable *z* of type **complex**:

```
z == creal(z) + cimag(z)*I
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

carg( ), *cimag*( ), *conj*( ), *cproj*( )

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

NAME

creat — create a new file or rewrite an existing one

SYNOPSIS

OH
```
#include <sys/stat.h>
#include <fcntl.h>

int creat(const char *path, mode_t mode);
```

DESCRIPTION

The *creat*( ) function shall behave as if it is implemented as follows:

```
int creat(const char *path, mode_t mode)
{
    return open(path, O_WRONLY|O_CREAT|O_TRUNC, mode);
}
```

RETURN VALUE

Refer to *open*( ).

ERRORS

Refer to *open*( ).

EXAMPLES

Creating a File

The following example creates the file **/tmp/file** with read and write permissions for the file owner and read permission for group and others. The resulting file descriptor is assigned to the *fd* variable.

```
#include <fcntl.h>
...
int fd;
mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
char *pathname = "/tmp/file";
...
fd = creat(pathname, mode);
...
```

APPLICATION USAGE

In multi-threaded applications, the *creat*( ) function can leak file descriptors into child processes. Applications should instead use *open*( ) with the O_CLOEXEC and O_CLOFORK flags to avoid the leak.

RATIONALE

The *creat*( ) function is redundant. Its services are also provided by the *open*( ) function. It has been included primarily for historical purposes since many existing applications depend on it. It is best considered a part of the C binding rather than a function that should be provided in other languages.

FUTURE DIRECTIONS

None.

SEE ALSO

mknod( ), *open*( )

XBD **<fcntl.h>**, **<sys/stat.h>**, **<sys/types.h>**

26920 **CHANGE HISTORY**

26921 First released in Issue 1. Derived from Issue 1 of the SVID.

26922 **Issue 6**

26923 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

26924 The following new requirements on POSIX implementations derive from alignment with the
26925 Single UNIX Specification:

26926 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
26927 required for conforming implementations of previous POSIX specifications, it was not
26928 required for UNIX applications.

26929 **Issue 7**

26930 SD5-XSH-ERN-186 is applied.

26931 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0064 [291] is applied.

26932 **Issue 8**

26933 Austin Group Defects 411 and 1318 are applied, changing the APPLICATION USAGE section.

NAME

crypt — password hashing function (**CRYPT**)

SYNOPSIS

XSI
```
#include <unistd.h>

char *crypt(const char *key, const char *salt);
```

DESCRIPTION

The *crypt*( ) function hashes a password for storage in the user database. The algorithm is implementation-defined.

The *key* argument points to a password to be hashed. The *salt* argument shall be a string of at least two bytes in length not including the null character chosen from the set:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 . /
```

The first two bytes of this string may be used to perturb the encoding algorithm.

The return value of *crypt*( ) points to static data that is overwritten by each call.

The *crypt*( ) function need not be thread-safe.

RETURN VALUE

Upon successful completion, *crypt*( ) shall return a pointer to the hashed password; the first two bytes of the returned value shall be those of the *salt* argument. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *crypt*( ) function shall fail if:

[ENOSYS] The functionality is not supported on this implementation.

EXAMPLES

Encoding Passwords

The following example finds a user database entry matching a particular user name and changes the current password to a new password. The *crypt*( ) function generates an encoded version of each password. The first call to *crypt*( ) produces an encoded version of the old password; that encoded password is then compared to the password stored in the user database. The second call to *crypt*( ) encodes the new password before it is stored.

The *putpwent*( ) function, used in the following example, is not part of POSIX.1-2024.

```
#include <unistd.h>
#include <pwd.h>
#include <string.h>
#include <stdio.h>
...
int valid_change;
int pfd;  /* Integer for file descriptor returned by open(). */
FILE *fpfd;  /* File pointer for use in putpwent(). */
struct passwd *p;
char user[100];
char oldpasswd[100];
char newpasswd[100];
```



```
        char savepasswd[100];
        ...
        valid_change = 0;
        while ((p = getpwent()) != NULL) {
            /* Change entry if found. */
            if (strcmp(p->pw_name, user) == 0) {
                if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
                    strcpy(savepasswd, crypt(newpasswd, user));
                    p->pw_passwd = savepasswd;
                    valid_change = 1;
                }
                else {
                    fprintf(stderr, "Old password is not valid\n");
                }
            }
            /* Put passwd entry into ptmp. */
            putpwent(p, fpfd);
        }
```

APPLICATION USAGE

The values returned by this function need not be portable among XSI-conformant systems.

Several implementations offer extensions via characters outside of the set specified for the *salt* argument for specifying alternative algorithms; while not portable, these extensions may offer better security. The use of *crypt*( ) for anything other than password hashing is not recommended.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

encrypt( ), *setkey*( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XSH-ERN-178 is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0073 [899] is applied.

Issue 8

Austin Group Defect 1192 is applied, clarifying that *crypt*( ) is intended for password hashing, not for general string encoding.

NAME

csin, csinf, csinl — complex sine functions

SYNOPSIS

```
#include <complex.h>

double complex csin(double complex z);
float complex csinf(float complex z);
long double complex csinl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the complex sine of *z*.

RETURN VALUE

These functions shall return the complex sine value.

MXC *csin*(*conj*(*iz*)), *csinf*(*conjf*(*iz*)), and *csinl*(*conjl*(*iz*)) shall return exactly the same value as *conj*(*csin*(*iz*)), *conjf*(*csinf*(*iz*)), and *conjl*(*csinl*(*iz*)), respectively, and *csin*(–*iz*), *csinf*(–*iz*), and *csinl*(–*iz*) shall return exactly the same value as –*csin*(*iz*), –*csinf*(*iz*), and –*csinl*(*iz*), respectively, including for the special values of *iz* below.

If *iz* is +0 + *i*0, –*i* (+0 + *i*0) shall be returned.

If *iz* is +0 + *i*Inf, –*i* (±0 + *i*NaN) shall be returned and the invalid floating-point exception shall be raised; the sign of the imaginary part of the result is unspecified.

If *iz* is +0 + *i*NaN, –*i* (±0 + *i*NaN) shall be returned; the sign of the imaginary part of the result is unspecified.

If *iz* is *x* + *i*Inf where *x* is positive and finite, –i (NaN + *i*NaN) shall be returned and the invalid floating-point exception shall be raised.

If *iz* is *x* + *i*NaN where *x* is non-zero and finite, –i (NaN + *i*NaN) shall be returned and the invalid floating-point exception may be raised.

If *iz* is +Inf + *i*0, –*i* (+Inf + *i*0) shall be returned.

If *iz* is +Inf + *iy* where *y* is positive and finite, –*i*Inf (*cos*(*y*) + *i sin*(*y*)) shall be returned.

If *iz* is +Inf + *i*Inf, –*i* (±Inf + *i*NaN) shall be returned and the invalid floating-point exception shall be raised; the sign of the imaginary part of the result is unspecified.

If *iz* is +Inf + *i*NaN, –*i* (±Inf + *i*NaN) shall be returned; the sign of the imaginary part of the result is unspecified.

If *iz* is NaN + *i*0, –*i* (NaN + *i*0) shall be returned.

If *iz* is NaN + *iy* where *y* is any non-zero number, –*i* (NaN + *i*NaN) shall be returned and the invalid floating-point exception may be raised.

If *iz* is NaN + *i*NaN, –*i* (NaN + *i*NaN) shall be returned.

ERRORS

No errors are defined.

27059 **EXAMPLES**

27060 None.

27061 **APPLICATION USAGE**

27062 None.

27063 **RATIONALE**

27064 The MXC special cases for *csin*( ) are derived from those for *csinh*( ) by applying the formula
27065 $csin(z) = -i\,csinh(iz)$.

27066 **FUTURE DIRECTIONS**

27067 None.

27068 **SEE ALSO**

27069 *casin*( ), *csinh*( )

27070 XBD **<complex.h>**

27071 **CHANGE HISTORY**

27072 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

27073 **Issue 8**

27074 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018
27075 standard.

NAME

csinh, csinhf, csinhl — complex hyperbolic sine functions

SYNOPSIS

```
#include <complex.h>

double complex csinh(double complex z);
float complex csinhf(float complex z);
long double complex csinhl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the complex hyperbolic sine of *z*.

RETURN VALUE

These functions shall return the complex hyperbolic sine value.

MXC *csinh*(*conj*(*z*)), *csinhf*(*conjf*(*z*)), and *csinhl*(*conjl*(*z*)) shall return exactly the same value as *conj*(*csinh*(*z*)), *conjf*(*csinhf*(*z*)), and *conjl*(*csinhl*(*z*)), respectively, and *csinh*(–*z*), *csinhf*(–*z*), and *csinhl*(–*z*) shall return exactly the same value as –*csinh*(*z*), –*csinhf*(*z*), and –*csinhl*(*z*), respectively, including for the special values of *z* below.

If *z* is +0 + *i*0, +0 + *i*0 shall be returned.

If *z* is +0 + *i*Inf, ±0 + *i*NaN shall be returned and the invalid floating-point exception shall be raised; the sign of the real part of the result is unspecified.

If *z* is +0 + *i*NaN, ±0 + *i*NaN shall be returned; the sign of the real part of the result is unspecified.

If *z* is *x* + *i*Inf where *x* is positive and finite, NaN + *i*NaN shall be returned and the invalid floating-point exception shall be raised.

If *z* is *x* + *i*NaN where *x* is non-zero and finite, NaN + *i*NaN shall be returned and the invalid floating-point exception may be raised.

If *z* is +Inf + *i*0, +Inf + *i*0 shall be returned.

If *z* is +Inf + *iy* where *y* is positive and finite, +Inf (*cos*(*y*) + *i sin*(*y*)) shall be returned.

If *z* is +Inf + *i*Inf, ±Inf + *i*NaN shall be returned and the invalid floating-point exception shall be raised; the sign of the real part of the result is unspecified.

If *z* is +Inf + *i*NaN, ±Inf + *i*NaN shall be returned; the sign of the real part of the result is unspecified.

If *z* is NaN + *i*0, NaN + *i*0 shall be returned.

If *z* is NaN + *iy* where *y* is any non-zero number, NaN + *i*NaN shall be returned and the invalid floating-point exception may be raised.

If *z* is NaN + *i*NaN, NaN + *i*NaN shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

casinh( )

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

27131 **NAME**

27132 csinl — complex sine functions

27133 **SYNOPSIS**

27134 `#include <complex.h>`

27135 `long double complex csinl(long double complex z);`

27136 **DESCRIPTION**

27137 Refer to *csin( )*.

NAME

csqrt, csqrtf, csqrtl — complex square root functions

SYNOPSIS

```
#include <complex.h>

double complex csqrt(double complex z);
float complex csqrtf(float complex z);
long double complex csqrtl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the complex square root of *z*, with a branch cut along the negative real axis.

RETURN VALUE

These functions shall return the complex square root value, in the range of the right half-plane (including the imaginary axis).

MXC *csqrt*(*conj*(z)), *csqrtf* (*conjf* (*z*)), and *csqrtl*(*conjl*(*z*)) shall return exactly the same value as *conj*(*csqrt*(z)), *conjf* (*csqrtf* (*z*)), and *conjl*(*csqrtl*(*z*)), respectively, including for the special values of *z* below.

If *z* is ±0 + *i*0, +0 + *i*0 shall be returned.

If the imaginary part of *z* is Inf, +Inf + *i*Inf, shall be returned.

If *z* is *x* + *i*NaN where *x* is finite, NaN + *i*NaN shall be returned and the invalid floating-point exception may be raised.

If *z* is –Inf + *iy* where *y* is positive-signed and finite, +0 + *i*Inf shall be returned.

If *z* is +Inf + *iy* where *y* is positive-signed and finite, +Inf + *i*0 shall be returned.

If *z* is –Inf + *i*NaN, NaN ± *i*Inf shall be returned; the sign of the imaginary part of the result is unspecified.

If *z* is +Inf + *i*NaN, +Inf + *i*NaN shall be returned.

If *z* is NaN + *iy* where *y* is finite, NaN + *i*NaN shall be returned and the invalid floating-point exception may be raised.

If *z* is NaN + *i*NaN, NaN + *i*NaN shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

27177 **FUTURE DIRECTIONS**

27178 None.

27179 **SEE ALSO**

27180 *cabs*( ), *cpow*( )

27181 XBD **<complex.h>**

27182 **CHANGE HISTORY**

27183 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

27184 **Issue 8**

27185 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018
27186 standard.

27187 **NAME**

27188 ctan, ctanf, ctanl — complex tangent functions

27189 **SYNOPSIS**

27190
```
#include <complex.h>
```

27191
```
double complex ctan(double complex z);
```
27192
```
float complex ctanf(float complex z);
```
27193
```
long double complex ctanl(long double complex z);
```

27194 **DESCRIPTION**

27195 CX The functionality described on this reference page is aligned with the ISO C standard. Any
27196 conflict between the requirements described here and the ISO C standard is unintentional. This
27197 volume of POSIX.1-2024 defers to the ISO C standard.

27198 These functions shall compute the complex tangent of *z*.

27199 **RETURN VALUE**

27200 These functions shall return the complex tangent value.

27201 MXC *ctan*(*conj*(*iz*)), *ctanf*(*conjf*(*iz*)), and *ctanl*(*conjl*(*iz*)) shall return exactly the same value as
27202 *conj*(*ctan*(*iz*)), *conjf*(*ctanf*(*iz*)), and *conjl*(*ctanl*(*iz*)), respectively, and *ctan*(–*iz*), *ctanf*(–*iz*), and
27203 *ctanl*(–*iz*) shall return exactly the same value as –*ctan*(*iz*), –*ctanf*(*iz*), and –*ctanl*(*iz*), respectively,
27204 including for the special values of *iz* below.

27205 If *iz* is +0 + *i*0, –*i* (+0 + *i*0) shall be returned.

27206 If *iz* is 0 + *i*Inf, –*i* (0 + *i*NaN) shall be returned and the invalid floating-point exception shall be
27207 raised.

27208 If *iz* is *x* + *i*Inf where *x* is non-zero and finite, –*i* (NaN + *i*NaN) shall be returned and the invalid
27209 floating-point exception shall be raised.

27210 If *iz* is 0 + *i*NaN, –*i* (0 + *i*NaN) shall be returned.

27211 If *iz* is *x* + *i*NaN where *x* is non-zero and finite, –*i* (NaN + *i*NaN) shall be returned and the
27212 invalid floating-point exception may be raised.

27213 If *iz* is +Inf + *iy* where *y* is positive-signed and finite, –*i* (1 + *i*0 *sin*(2*y*)) shall be returned.

27214 If *iz* is +Inf + *i*Inf, –*i* (1 ± *i*0) shall be returned; the sign of the real part of the result is unspecified.

27215 If *iz* is +Inf + *i*NaN, –*i* (1 ± *i*0) shall be returned; the sign of the real part of the result is
27216 unspecified.

27217 If *iz* is NaN + *i*0, –*i* (NaN + *i*0) shall be returned.

27218 If *iz* is NaN + *iy* where *y* is any non-zero number, –*i* (NaN + *i*NaN) shall be returned and the
27219 invalid floating-point exception may be raised.

27220 If *iz* is NaN + *i*NaN, –*i* (NaN + *i*NaN) shall be returned.

27221 **ERRORS**

27222 No errors are defined.

27223 **EXAMPLES**

27224 None.

27225 **APPLICATION USAGE**

27226 None.

27227 **RATIONALE**

27228 The MXC special cases for *ctan*( ) are derived from those for *ctanh*( ) by applying the formula
27229 $ctan(z) = -i\,ctanh(iz)$.

27230 **FUTURE DIRECTIONS**

27231 None.

27232 **SEE ALSO**

27233 *catan*( ), *ctanh*( )

27234 XBD **<complex.h>**

27235 **CHANGE HISTORY**

27236 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

27237 **Issue 8**

27238 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018
27239 standard.

27240 **NAME**

27241 ctanh, ctanhf, ctanhl — complex hyperbolic tangent functions

27242 **SYNOPSIS**

27243
```
#include <complex.h>
```

27244
```
double complex ctanh(double complex z);
```
27245
```
float complex ctanhf(float complex z);
```
27246
```
long double complex ctanhl(long double complex z);
```

27247 **DESCRIPTION**

27248 CX The functionality described on this reference page is aligned with the ISO C standard. Any
27249 conflict between the requirements described here and the ISO C standard is unintentional. This
27250 volume of POSIX.1-2024 defers to the ISO C standard.

27251 These functions shall compute the complex hyperbolic tangent of *z*.

27252 **RETURN VALUE**

27253 These functions shall return the complex hyperbolic tangent value.

27254 MXC *ctanh*(*conj*(*z*)), *ctanhf* (*conjf* (*z*)), and *ctanhl*(*conjl*(*z*)) shall return exactly the same value as
27255 *conj*(*ctanh*(*z*)), *conjf* (*ctanhf*(z)), and *conjl*(*ctanhl*(*z*)), respectively, and *ctanh*(–*z*), *ctanhf* (–*z*), and
27256 *ctanhl*(–*z*) shall return exactly the same value as –*ctanh*(*z*), –*ctanhf* (*z*), and –*ctanhl*(*z*),
27257 respectively, including for the special values of *z* below.

27258 If *z* is +0 + *i*0, +0 + *i*0 shall be returned.

27259 If *z* is 0 + *i*Inf, 0 + *i*NaN shall be returned and the invalid floating-point exception shall be raised.

27260 If *z* is *x* + *i*Inf where *x* is non-zero and finite, NaN + *i*NaN shall be returned and the invalid
27261 floating-point exception shall be raised.

27262 If *z* is 0 + *i*NaN, 0 + *i*NaN shall be returned.

27263 If *z* is *x* + *i*NaN where *x* is non-zero and finite, NaN + *i*NaN shall be returned and the invalid
27264 floating-point exception may be raised.

27265 If *z* is +Inf + *iy* where *y* is positive-signed and finite, 1 + *i*0 *sin*(2*y*) shall be returned.

27266 If *z* is +Inf + *i*Inf, 1 ± *i*0 shall be returned; the sign of the imaginary part of the result is
27267 unspecified.

27268 If *z* is +Inf + *i*NaN, 1 ± *i*0 shall be returned; the sign of the imaginary part of the result is
27269 unspecified.

27270 If *z* is NaN + *i*0, NaN + *i*0 shall be returned.

27271 If *z* is NaN + *iy* where *y* is any non-zero number, NaN + *i*NaN shall be returned and the invalid
27272 floating-point exception may be raised.

27273 If *z* is NaN + *i*NaN, NaN + *i*NaN shall be returned.

27274 **ERRORS**

27275 No errors are defined.

27276 **EXAMPLES**

27277 None.

27278 **APPLICATION USAGE**

27279 None.

27280 **RATIONALE**

27281 None.

27282 **FUTURE DIRECTIONS**

27283 None.

27284 **SEE ALSO**

27285 *catanh( )*

27286 XBD **<complex.h>**

27287 **CHANGE HISTORY**

27288 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

27289 **Issue 8**

27290 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018
27291 standard.

27292 **NAME**

27293 ctanl — complex tangent functions

27294 **SYNOPSIS**

27295 `#include <complex.h>`

27296 `long double complex ctanl(long double complex z);`

27297 **DESCRIPTION**

27298 Refer to *ctan( )*.

NAME

ctermid — generate a pathname for the controlling terminal

SYNOPSIS

CX
```
#include <stdio.h>

char *ctermid(char *s);
```

DESCRIPTION

The *ctermid*( ) function shall generate a string that, when used as a pathname, refers to the current controlling terminal for the current process. If *ctermid*( ) returns a pathname, access to the file is not guaranteed.

The *ctermid*( ) function need not be thread-safe if called with a NULL parameter.

RETURN VALUE

If *s* is a null pointer, the string shall be generated in an area that may be static, the address of which shall be returned. The application shall not modify the string returned. The returned pointer might be invalidated or the string content might be overwritten by a subsequent call to *ctermid*( ). The returned pointer might also be invalidated if the calling thread is terminated. If *s* is not a null pointer, *s* is assumed to point to a character array of at least L_ctermid bytes; the string is placed in this array and the value of *s* shall be returned. The symbolic constant L_ctermid is defined in **<stdio.h>**, and shall have a value greater than 0.

The *ctermid*( ) function shall return an empty string if the pathname that would refer to the controlling terminal cannot be determined, or if the function is unsuccessful.

ERRORS

No errors are defined.

EXAMPLES

Determining the Controlling Terminal for the Current Process

The following example returns a pointer to a string that identifies the controlling terminal for the current process. The pathname for the terminal is stored in the array pointed to by the *ptr* argument, which has a size of L_ctermid bytes, as indicated by the *term* argument.

```
#include <stdio.h>
...
char term[L_ctermid];
char *ptr;

ptr = ctermid(term);
```

APPLICATION USAGE

The difference between *ctermid*( ) and *ttyname*( ) is that *ttyname*( ) must be handed a file descriptor and return a path of the terminal associated with that file descriptor, while *ctermid*( ) returns a string (such as `"/dev/tty"`) that refers to the current controlling terminal if used as a pathname.

RATIONALE

L_ctermid must be defined appropriately for a given implementation and must be greater than zero so that array declarations using it are accepted by the compiler. The value includes the terminating null byte.

Conforming applications that use multiple threads cannot call *ctermid*( ) with NULL as the parameter. If *s* is not NULL, the *ctermid*( ) function generates a string that, when used as a

pathname, refers to the current controlling terminal for the current process. If *s* is NULL, the return value of *ctermid*( ) is undefined.

There is no additional burden on the programmer—changing to use a hypothetical thread-safe version of *ctermid*( ) along with allocating a buffer is more of a burden than merely allocating a buffer. Application code should not assume that the returned string is short, as some implementations have more than two pathname components before reaching a logical device name.

FUTURE DIRECTIONS

None.

SEE ALSO

ttyname( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #148 is applied, updating the RATIONALE.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0065 [75,428] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0074 [656] is applied.

NAME

ctime — convert a time value to a date and time string

SYNOPSIS

OB
```
#include <time.h>

char *ctime(const time_t *clock);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *ctime*( ) function shall convert the time pointed to by *clock*, representing time in seconds since the Epoch, to local time in the form of a string. It shall be equivalent to:

```
asctime(localtime(clock))
```

The *asctime*( ), *ctime*( ), *gmtime*( ), and *localtime*( ) functions shall return values in one of two static objects: a broken-down time structure and an array of **char**. Execution of any of the functions that return a pointer to one of these object types may overwrite the information in any object of the same type pointed to by the value returned from any previous call to any of them.

The *ctime*( ) function need not be thread-safe; however, *ctime*( ) shall avoid data races with all functions other than itself, *asctime*( ), *gmtime*( ), and *localtime*( ).

RETURN VALUE

The *ctime*( ) function shall return the pointer returned by *asctime*( ) with that broken-down time as an argument.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

This function is included only for compatibility with older implementations. It has undefined behavior if the resulting string would be too long, so the use of this function should be discouraged. On implementations that do not detect output string length overflow, it is possible to overflow the output buffer in such a way as to cause applications to fail, or possible system security violations. Also, this function does not support localized date and time formats. To avoid these problems, applications should use *strftime*( ) to generate strings from broken-down times.

Values for the broken-down time structure can be obtained by calling *gmtime*( ) or *localtime*( ).

Attempts to use *ctime*( ) for times before the Epoch or for times beyond the year 9999 produce undefined results. Refer to *asctime*( ) (on page 617).

RATIONALE

The standard developers decided to mark the *ctime*( ) function obsolescent even though it is in the ISO C standard due to the possibility of buffer overflow. The ISO C standard also provides the *strftime*( ) function which can be used to avoid these problems.

FUTURE DIRECTIONS

This function may be removed in a future version, but not until after it has been removed from the ISO C standard.

SEE ALSO

asctime( ), *clock*( ), *difftime*( ), *futimens*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( )

XBD **<time.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

The *ctime_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *ctime*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

Extensions beyond the ISO C standard are marked.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XSH-ERN-25 is applied, updating the APPLICATION USAGE.

Austin Group Interpretation 1003.1-2001 #053 is applied, marking these functions obsolescent.

The *ctime_r*( ) function is moved from the Thread-Safe Functions option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0066 [321,428] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0075 [664] is applied.

Issue 8

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018 standard.

Austin Group Defect 1330 is applied, changing the FUTURE DIRECTIONS section.

Austin Group Defect 1376 is applied, removing CX shading from some text derived from the ISO C standard and updating it to match the ISO C standard.

Austin Group Defect 1410 is applied, removing the *ctime_r*( ) function.

NAME

daylight — daylight saving time flag

SYNOPSIS

XSI

```
#include <time.h>

extern int daylight;
```

DESCRIPTION

Refer to *tzset*( ).

NAME

dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey, dbm_open, dbm_store — database functions

SYNOPSIS

XSI

```
#include <ndbm.h>

int dbm_clearerr(DBM *db);
void dbm_close(DBM *db);
int dbm_delete(DBM *db, datum key);
int dbm_error(DBM *db);
datum dbm_fetch(DBM *db, datum key);
datum dbm_firstkey(DBM *db);
datum dbm_nextkey(DBM *db);
DBM *dbm_open(const char *file, int open_flags, mode_t file_mode);
int dbm_store(DBM *db, datum key, datum content, int store_mode);
```

DESCRIPTION

These functions create, access, and modify a database.

A **datum** consists of at least two members, *dptr* and *dsize*. The *dptr* member points to an object that is *dsize* bytes in length. Arbitrary binary data, as well as character strings, may be stored in the object pointed to by *dptr*.

A database shall be stored in one or two files. When one file is used, the name of the database file shall be formed by appending the suffix **.db** to the *file* argument given to *dbm_open*( ). When two files are used, the names of the database files shall be formed by appending the suffixes **.dir** and **.pag** respectively to the *file* argument.

The *dbm_open*( ) function shall open a database. The *file* argument to the function is the pathname of the database. Values for the *open_flags* argument are constructed by a bitwise-inclusive OR of flags from the following list, defined in **<fcntl.h>**. Applications shall specify exactly one of the first three values (file access modes) below in the value of *open_flags*:

O_RDONLY — Open the database, and the underlying file(s) used to store the database, for reading only.

O_RDWR — Open the database, and the underlying file(s) used to store the database, for reading and writing.

O_WRONLY — Open the database for writing only. The underlying file(s) used to store the database shall be opened for reading and writing.

Any combination of the following can be used:

O_CLOEXEC — If set, the FD_CLOEXEC flag for the new file descriptor(s) used to open the underlying file(s) shall be set.

O_CREAT — If the database exists, this flag has no effect except as noted under O_EXCL below. Otherwise, the database shall be created; the user ID of the underlying file(s) shall be set to the effective user ID of the process; the group ID of the underlying file(s) shall be set to the group ID of the file's parent directory or to the effective group ID of the process. Implementations shall provide a way to initialize the group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the group ID to the effective group ID of the calling process.

SIO O_DSYNC Write I/O operations on the file(s) used to store the database shall complete as defined by synchronized I/O data integrity completion.

O_EXCL If O_CREAT and O_EXCL are set, *dbm_open*( ) shall fail if the database exists according to the following criteria. If the database is stored in a file with a **.db** suffix, a check for the existence of the file and the creation of the file if it does not exist shall be performed as a single atomic operation. If the database is stored in files with **.pag** and **.dir** suffixes, this atomic existence check and creation operation shall be performed for each file, but it is unspecified which file is first. If the first file is successfully created and the second file is found to exist, the first file should be removed before *dbm_open*( ) returns.

If O_EXCL is set and O_CREAT is not set, the result is undefined.

SIO O_RSYNC Read I/O operations on the file(s) used to store the database shall complete at the same level of integrity as specified by the O_DSYNC and O_SYNC flags. If both O_DSYNC and O_RSYNC are set in oflag, all I/O operations on the file(s) shall complete as defined by synchronized I/O data integrity completion. If both O_SYNC and O_RSYNC are set in flags, all I/O operations on the file(s) shall complete as defined by synchronized I/O file integrity completion.

O_SYNC Write I/O operations on the file(s) used to store the database shall complete as defined by synchronized I/O file integrity completion.

O_TRUNC If the database exists and is successfully opened O_RDWR or O_WRONLY, it shall be emptied, and the mode and owner of the underlying file(s) shall be unchanged. The result of using O_TRUNC without either O_RDWR or O_WRONLY is undefined.

The behaviour of other flags described for the *flags* argument of *open*( ) is unspecified.

The *file_mode* argument has the same meaning as the third argument of *open*( ) and shall apply to the underlying file(s) used to store the database.

The *dbm_open*( ) function need not accept pathnames longer than {PATH_MAX}−4 bytes (including the terminating null), or pathnames with a last component longer than {NAME_MAX}−4 bytes (excluding the terminating null).

The *dbm_close*( ) function shall close a database. The application shall ensure that argument *db* is a pointer to a **dbm** structure that has been returned from a call to *dbm_open*( ).

These database functions shall support an internal block size large enough to support key/content pairs of at least 1 023 bytes.

The *dbm_fetch*( ) function shall read a record from a database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*( ). The argument *key* is a **datum** that has been initialized by the application to the value of the key that matches the key of the record the program is fetching.

The *dbm_store*( ) function shall write a record to a database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*( ). The argument *key* is a **datum** that has been initialized by the application to the value of the key that identifies (for subsequent reading, writing, or deleting) the record the application is writing. The argument *content* is a **datum** that has been initialized by the application to the value of the record the program is writing. The argument *store_mode* controls whether *dbm_store*( ) replaces any pre-existing record that has the same key that is specified by the *key* argument. The application shall

set *store_mode* to either DBM_INSERT or DBM_REPLACE. If the database contains a record that matches the *key* argument and *store_mode* is DBM_REPLACE, the existing record shall be replaced with the new record. If the database contains a record that matches the *key* argument and *store_mode* is DBM_INSERT, the existing record shall be left unchanged and the new record ignored. If the database does not contain a record that matches the *key* argument and *store_mode* is either DBM_INSERT or DBM_REPLACE, the new record shall be inserted in the database.

If the sum of a key/content pair exceeds the internal block size, the result is unspecified. Moreover, the application shall ensure that all key/content pairs that hash together fit on a single block. The *dbm_store*( ) function shall return an error in the event that a disk block fills with inseparable data.

The *dbm_delete*( ) function shall delete a record and its key from the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*( ). The argument *key* is a **datum** that has been initialized by the application to the value of the key that identifies the record the program is deleting.

The *dbm_firstkey*( ) function shall return the first key in the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*( ).

The *dbm_nextkey*( ) function shall return the next key in the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*( ). The application shall ensure that the *dbm_firstkey*( ) function is called before calling *dbm_nextkey*( ). Subsequent calls to *dbm_nextkey*( ) return the next key until all of the keys in the database have been returned.

The *dbm_error*( ) function shall return the error condition of the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*( ).

The *dbm_clearerr*( ) function shall clear the error condition of the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*( ).

The *dptr* pointers returned by these functions may point into static storage that may be changed by subsequent calls.

These functions need not be thread-safe.

RETURN VALUE

The *dbm_store*( ) and *dbm_delete*( ) functions shall return 0 when they succeed and a negative value when they fail.

The *dbm_store*( ) function shall return 1 if it is called with a *flags* value of DBM_INSERT and the function finds an existing record with the same key.

The *dbm_error*( ) function shall return 0 if the error condition is not set and return a non-zero value if the error condition is set.

The return value of *dbm_clearerr*( ) is unspecified.

The *dbm_firstkey*( ) and *dbm_nextkey*( ) functions shall return a key **datum**. When the end of the database is reached, the *dptr* member of the key is a null pointer. If an error is detected, the *dptr* member of the key shall be a null pointer and the error condition of the database shall be set.

The *dbm_fetch*( ) function shall return a content **datum**. If no record in the database matches the key or if an error condition has been detected in the database, the *dptr* member of the content shall be a null pointer.

The *dbm_open*( ) function shall return a pointer to a database structure. If an error is detected during the operation, *dbm_open*( ) shall return a (**DBM** ***)0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The following code can be used to traverse the database:

```
for(key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

The *dbm_** functions provided in this library should not be confused in any way with those of a general-purpose database management system. These functions do not provide for multiple search keys per entry, they do not protect against multi-user access (in other words they do not lock records or files), and they do not provide the many other useful database functions that are found in more robust database management systems. Creating and updating databases by use of these functions is relatively slow because of data copies that occur upon hash collisions. These functions are useful for applications requiring fast lookup of relatively static information that is to be indexed by a single key.

Note that a strictly conforming application is extremely limited by these functions: since there is no way to determine that the keys in use do not all hash to the same value (although that would be rare), a strictly conforming application cannot be guaranteed that it can store more than one block's worth of data in the database. As long as a key collision does not occur, additional data may be stored, but because there is no way to determine whether an error is due to a key collision or some other error condition (*dbm_error*( ) being effectively a Boolean), once an error is detected, the application is effectively limited to guessing what the error might be if it wishes to continue using these functions.

The *dbm_delete*( ) function need not physically reclaim file space, although it does make it available for reuse by the database.

After calling *dbm_store*( ) or *dbm_delete*( ) during a pass through the keys by *dbm_firstkey*( ) and *dbm_nextkey*( ), the application should reset the database by calling *dbm_firstkey*( ) before again calling *dbm_nextkey*( ). The contents of these files are unspecified and may not be portable.

Applications should take care that database pathname arguments specified to *dbm_open*( ) are not prefixes of unrelated files. This might be done, for example, by placing databases in a separate directory.

Since some implementations use three characters for a suffix and others use four characters for a suffix, applications should ensure that the maximum portable pathname length passed to *dbm_open*( ) is no greater than {PATH_MAX}–4 bytes, with the last component of the pathname no greater than {NAME_MAX}–4 bytes.

RATIONALE

Previously the standard required the database to be stored in two files, one file being a directory containing a bitmap of keys and having **.dir** as its suffix. The second file containing all data and having **.pag** as its suffix. This has been changed not to specify the use of the files and to allow newer implementations of the Berkeley DB interface using a single file that have evolved while remaining compatible with the application programming interface. The standard developers considered removing the specific suffixes altogether but decided to retain them so as not to pollute the application file name space more than necessary and to allow for portable backups of the database.

27623 **FUTURE DIRECTIONS**

27624 A future revision of this standard may mandate the file removal that is currently recommended
27625 (by the use of ``should'') in the description of O_EXCL.

27626 **SEE ALSO**

27627 *open*( )

27628 XBD **<fcntl.h>**, **<ndbm.h>**

27629 **CHANGE HISTORY**

27630 First released in Issue 4, Version 2.

27631 **Issue 5**

27632 Moved from X/OPEN UNIX extension to BASE.

27633 Normative text previously in the APPLICATION USAGE section is moved to the
27634 DESCRIPTION.

27635 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

27636 **Issue 6**

27637 The normative text is updated to avoid use of the term ``must'' for application requirements.

27638 **Issue 7**

27639 Austin Group Interpretation 1003.1-2001 #042 is applied so that the DESCRIPTION permits
27640 newer implementations of the Berkeley DB interface.

27641 Austin Group Interpretation 1003.1-2001 #156 is applied.

27642 **Issue 8**

27643 Austin Group Defect 1057 is applied, clarifying how the O_ flags defined for use with *open*( )
27644 apply to *dbm_open*( ).

NAME

dcgettext, dcgettext_l, dcngettext, dcngettext_l, dgettext, dgettext_l — message handling functions

SYNOPSIS

```
#include <libintl.h>

char *dcgettext(const char *domainname, const char *msgid,
    int category);
char *dcgettext_l(const char *domainname, const char *msgid,
    int category, locale_t locale);
char *dcngettext(const char *domainname, const char *msgid,
    const char *msgid_plural, unsigned long int n,
    int category);
char *dcngettext_l(const char *domainname, const char *msgid,
    const char *msgid_plural, unsigned long int n,
    int category, locale_t locale);
char *dgettext(const char *domainname, const char *msgid);
char *dgettext_l(const char *domainname, const char *msgid,
    locale_t locale);
```

DESCRIPTION

Refer to *gettext*.

NAME

difftime — compute the difference between two calendar time values

SYNOPSIS

```
#include <time.h>

double difftime(time_t time1, time_t time0);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *difftime*( ) function shall compute the difference between two calendar times (as returned by *time*( )): *time*1– *time*0.

RETURN VALUE

The *difftime*( ) function shall return the difference expressed in seconds as a type **double**.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime( ), *clock*( ), *ctime*( ), *futimens*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( )

XBD **<time.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ISO C standard.

NAME

dirfd — extract the file descriptor used by a DIR stream

SYNOPSIS

```
#include <dirent.h>

int dirfd(DIR *dirp);
```

DESCRIPTION

If the directory stream referenced by *dirp* has an associated file descriptor, *dirfd*( ) shall return that file descriptor. Otherwise, *dirfd*( ) shall open a new file description referring to the directory associated with the directory stream as if by calling:

```
open(DirectoryName, O_RDONLY | O_DIRECTORY | O_CLOEXEC);
```

except that no pathname for use as *DirectoryName* need exist or be accessible. It shall then associate the new file descriptor with the directory stream, and return that file descriptor.

Upon successful return from *dirfd*( ), the file descriptor is under the control of the system, and if any attempt is made to close the file descriptor, or to modify the state of the associated description, other than by means of *closedir*( ), *readdir*( ), *readdir_r*( ), *rewinddir*( ), or *seekdir*( ), the behavior is undefined. Upon calling *closedir*( ) the file descriptor shall be closed.

XSI

RETURN VALUE

Upon successful completion, the *dirfd*( ) function shall return an integer which contains a file descriptor for the stream pointed to by *dirp*. Otherwise, it shall return –1 and shall set *errno* to indicate the error.

ERRORS

The *dirfd*( ) function shall fail if:

[EMFILE] A new file descriptor is required and all file descriptors available to the process are currently open.

[ENFILE] A new file descriptor is required and the maximum allowable number of files is currently open in the system.

The *dirfd*( ) function may fail if:

[EINVAL] The *dirp* argument does not refer to a valid directory stream.

EXAMPLES

None.

APPLICATION USAGE

The *dirfd*( ) function is intended to be a mechanism by which an application may obtain a file descriptor to use for the *fchdir*( ) function.

RATIONALE

This interface was introduced because the Base Definitions volume of POSIX.1-2024 does not make public the **DIR** data structure. Applications tend to use the *fchdir*( ) function on the file descriptor returned by this interface, and this has proven useful for security reasons; in particular, it is a better technique than others where directory names might change.

On an implementation where reading from a directory stream does not use a file descriptor, *opendir*( ) need not allocate one to be returned by *dirfd*( ). The implementation can instead delay the allocation of a suitable file descriptor until the first time *dirfd*( ) is called for the stream. A file descriptor allocated by *dirfd*( ) must be closed by *closedir*( ).

27735 **FUTURE DIRECTIONS**

27736 None.

27737 **SEE ALSO**

27738 *closedir*( ), *fchdir*( ), *fdopendir*( ), *fileno*( ), *open*( ), *readdir*( )

27739 XBD **<dirent.h>**

27740 **CHANGE HISTORY**

27741 First released in Issue 7.

27742 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0067 [422] is applied.

27743 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0076 [572] is applied.

27744 **Issue 8**

27745 Austin Group Defects 391 and 1359 are applied, changing the DESCRIPTION and RATIONALE
27746 sections, and adding the [EMFILE] and [ENFILE] errors.

NAME

dirname — report the parent directory name of a file pathname

SYNOPSIS

XSI
```
#include <libgen.h>

char *dirname(char *path);
```

DESCRIPTION

The *dirname*( ) function shall take a pointer to a character string that contains a pathname, and return a pointer to a string that is a pathname of the directory containing the entry of the final pathname component. The *dirname*( ) function shall not perform pathname resolution; the result shall not be affected by whether or not *path* exists or by its file type. Trailing '/' characters in the pathname that are not also leading '/' characters shall not be counted as part of the pathname.

If the pathname does not contain a '/', then *dirname*( ) shall return a pointer to the string ".". If *path* is a null pointer or points to an empty string, *dirname*( ) shall return a pointer to the string ".".

It is unspecified whether redundant '/' characters and '.' pathname components in *path* are removed after determining the pathname to output. However, ".." pathname components occurring prior to the final component shall not be removed.

The *dirname*( ) function may modify the string pointed to by *path*, and may return a pointer into the input string. The returned pointer might be invalidated if the input string is subsequently modified or freed. If *path* does not contain a '/', is a null pointer, or points to an empty string the returned pointer may point to constant data that cannot be modified.

RETURN VALUE

The *dirname*( ) function shall return a pointer to a string as described above.

The *dirname*( ) function shall always be successful and no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

The following code fragment reads a pathname, changes the current working directory to the parent directory, and opens the file.

```
char *path = NULL, *pathcopy;
size_t buflen = 0;
ssize_t linelen = 0;
int fd;

linelen = getline(&path, &buflen, stdin);

path[linelen-1] = 0;
pathcopy = strdup(path);
if (chdir(dirname(pathcopy)) < 0) {
    ...
}
if ((fd = open(basename(path), O_RDONLY)) >= 0) {
    ...
    close (fd);
}
```



```
...
free (pathcopy);
free (path);
```

The EXAMPLES section of the *basename*( ) function (see *basename*( )) includes a table showing examples of the results of processing several sample pathnames by the *basename*( ) and *dirname*( ) functions and by the *basename* and *dirname* utilities.

APPLICATION USAGE

The *dirname*( ) and *basename*( ) functions together yield a complete pathname. The expression *dirname*(*path*) obtains the pathname of the directory where *basename*(*path*) is found.

Since the meaning of the leading `"//"` is implementation-defined, *dirname*("**//foo**") may return either `"//"` or `"/"` (but nothing else).

Note that in some circumstances, the returned pointer might point into constant data. Therefore, if the application needs to modify the returned data, it should be copied first.

RATIONALE

An implementation should prefer the shortest output possible; however, this is not required, in part because earlier versions of the standard did not mention whether elision of redundant <slash> characters or dot (`"."`) components was permitted. Removal of the dot-dot (`".."`) pathname component is not permitted, because eliding it correctly would require performing pathname resolution to ensure the resulting string would still point to the correct pathname if the original string resolved as a pathname. On implementations where pathname `"//"` has an implementation-defined meaning distinct from the pathname `"/"`, the dirname of `"//"` will be `"//"`.

Earlier versions of this standard seemed to allow thread-safe and non-thread-safe implementations of *basename*( ) and *dirname*( ), but did not allow implementations to return a null pointer and require that *errno* be set when that happened. The standard now requires thread-safe behavior for both of these functions and clearly states that they are always successful.

FUTURE DIRECTIONS

None.

SEE ALSO

basename( )

XBD **<libgen.h>**

XCU *basename*, *dirname*

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

The EXAMPLES section is revised.

27835 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0068 [75] is applied.

27836 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0077 [830], XSH/TC2-2008/0078 [612],
27837 XSH/TC2-2008/0079 [830], XSH/TC2-2008/0080 [656], and XSH/TC2-2008/0081 [612] are
27838 applied.

27839 **Issue 8**

27840 Austin Group Defect 1064 is applied, requiring *dirname*( ) to be thread-safe and allowing it to
27841 return a pointer to constant data under certain conditions.

27842 Austin Group Defect 1073 is applied, changing ``parent directory of that file'' to ``directory
27843 containing the entry of the final pathname component'' and clarifying that redundant `'/'`
27844 characters and `'.'` pathname components may be removed after determining the pathname to
27845 output.

27846 **NAME**

27847 div — compute the quotient and remainder of an integer division

27848 **SYNOPSIS**

27849
```
#include <stdlib.h>
```

27850
```
div_t div(int numer, int denom);
```

27851 **DESCRIPTION**

27852 CX The functionality described on this reference page is aligned with the ISO C standard. Any
27853 conflict between the requirements described here and the ISO C standard is unintentional. This
27854 volume of POSIX.1-2024 defers to the ISO C standard.

27855 The *div*( ) function shall compute the quotient and remainder of the division of the numerator
27856 *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the integer
27857 of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be
27858 represented, the behavior is undefined; otherwise, *quot*denom+rem* shall equal *numer*.

27859 **RETURN VALUE**

27860 The *div*( ) function shall return a structure of type **div_t**, comprising both the quotient and the
27861 remainder. The structure includes the following members, in any order:

27862
```
int  quot;  /* quotient */
```
27863
```
int  rem;   /* remainder */
```

27864 **ERRORS**

27865 No errors are defined.

27866 **EXAMPLES**

27867 None.

27868 **APPLICATION USAGE**

27869 None.

27870 **RATIONALE**

27871 None.

27872 **FUTURE DIRECTIONS**

27873 None.

27874 **SEE ALSO**

27875 *ldiv*( )

27876 XBD **<stdlib.h>**

27877 **CHANGE HISTORY**

27878 First released in Issue 4. Derived from the ISO C standard.

NAME

dladdr — get information relating to an address

SYNOPSIS

```
#include <dlfcn.h>

int dladdr(const void *restrict addr, Dl_info_t *restrict dlip);
```

DESCRIPTION

The *dladdr*( ) function shall determine whether the address specified by *addr* is located within the address range occupied by a mapped object. The mapped objects examined shall include any executable object files that have previously been loaded by a call to *dlopen*( ) and for which *dlclose*( ) has not subsequently been called, and any shared library files that were loaded as dependencies of the executable file from which the current process image was loaded; they may also include any executable object files that have previously been loaded by a call to *dlopen*( ) and for which *dlclose*( ) has subsequently been called, the executable file from which the current process image was loaded, and implementation-defined additional mapped objects (for example, all regular files mapped using *mmap*( ) might be included). If the specified address is within the mapped address range of one of these mapped objects and the object contains a symbol table, the symbol table shall be searched for a symbol (a function identifier or a data object identifier) that has the largest address less than or equal to the specified address.

If the address specified by *addr* is within the mapped address range of one of the examined mapped objects, the structure pointed to by *dlip* shall be populated as follows:

- The value of the *dli_fname* member shall be set to point to the pathname of the mapped object. (This might no longer resolve to the file that was mapped, for example if it was a link that has subsequently been removed or renamed.)
- The value of the *dli_fbase* member shall be set to the base of the address range occupied by the mapped object.
- The value of the *dli_sname* member shall be set to point to the name of the symbol that has the largest address less than or equal to the specified address, or to a null pointer if no such symbol was found.
- If *dli_sname* is set to a null pointer, the value of the *dli_saddr* member shall also be set to a null pointer. Otherwise, if *dli_sname* names a function identifier, *dli_saddr* shall be set to the address of the function converted from type pointer to function to type pointer to **void**; otherwise, *dli_saddr* shall be set to the address of the data object named by *dli_sname* converted from a pointer to the type of the data object to a pointer to **void**.

RETURN VALUE

Upon successful completion, a non-zero value shall be returned. If the specified address is not located within the address range occupied by an examined mapped object, or if an error occurs, zero shall be returned. More detailed diagnostic information shall be available through *dlerror*( ).

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The **Dl_info_t** members may point to addresses within the mapped object. These pointers can become invalid if the object is unmapped (for example, loaded executable objects may be unloaded by *dlclose*( )).

If *dli_sname* names a function identifier, the value of *dli_saddr* can be converted back to type pointer to function using a cast in the manner shown in the *dlsym*( ) EXAMPLES section. Note that this conversion is not defined by the ISO C standard. This standard requires this conversion to work correctly on conforming implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

dlclose( ), *dlerror*( ), *dlopen*( ), *dlsym*( )

XBD **<dlfcn.h>**

CHANGE HISTORY

First released in Issue 8.

27937 **NAME**

27938 dlclose — close a symbol table handle

27939 **SYNOPSIS**

27940
```
#include <dlfcn.h>
```

27941
```
int dlclose(void *handle);
```

27942 **DESCRIPTION**

27943 The *dlclose*( ) function shall inform the system that the symbol table handle specified by *handle* is
27944 no longer needed by the application.

27945 An application writer may use *dlclose*( ) to make a statement of intent on the part of the process,
27946 but this statement does not create any requirement upon the implementation. When the symbol
27947 table handle is closed, the implementation may unload the executable object files that were
27948 loaded by *dlopen*( ) when the symbol table handle was opened and those that were loaded by
27949 *dlsym*( ) when using the symbol table handle identified by *handle*.

27950 Once a symbol table handle has been closed, an application should assume that any symbols
27951 (function identifiers and data object identifiers) made visible using *handle*, are no longer
27952 available to the process.

27953 Although a *dlclose*( ) operation is not required to remove any functions or data objects from the
27954 address space, neither is an implementation prohibited from doing so. The only restriction on
27955 such a removal is that no function nor data object shall be removed to which references have
27956 been relocated, until or unless all such references are removed. For instance, an executable object
27957 file that had been loaded with a *dlopen*( ) operation specifying the RTLD_GLOBAL flag might
27958 provide a target for dynamic relocations performed in the processing of other relocatable
27959 objects—in such environments, an application may assume that no relocation, once made, shall
27960 be undone or remade unless the executable object file containing the relocated object has itself
27961 been removed.

27962 **RETURN VALUE**

27963 If the referenced symbol table handle was successfully closed, *dlclose*( ) shall return 0. If *handle*
27964 does not refer to an open symbol table handle or if the symbol table handle could not be closed,
27965 *dlclose*( ) shall return a non-zero value. More detailed diagnostic information shall be available
27966 through *dlerror*( ).

27967 **ERRORS**

27968 No errors are defined.

27969 **EXAMPLES**

27970 The following example illustrates use of *dlopen*( ) and *dlclose*( ):

```
27971 #include <dlfcn.h>
27972 int eret;
27973 void *mylib;
27974 ...
27975 /* Open a dynamic library and then close it ... */
27976 mylib = dlopen("mylib.so", RTLD_LOCAL | RTLD_LAZY);
27977 ...
27978 eret = dlclose(mylib);
27979 ...
```


APPLICATION USAGE

A conforming application should employ a symbol table handle returned from a *dlopen*( ) invocation only within a given scope bracketed by a *dlopen*( ) operation and the corresponding *dlclose*( ) operation. Implementations are free to use reference counting or other techniques such that multiple calls to *dlopen*( ) referencing the same executable object file may return a pointer to the same data object as the symbol table handle.

Implementations are also free to re-use a handle. For these reasons, the value of a handle must be treated as an opaque data type by the application, used only in calls to *dlsym*( ) and *dlclose*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

dladdr( ), *dlerror*( ), *dlopen*( ), *dlsym*( )

XBD **<dlfcn.h>**

CHANGE HISTORY

First released in Issue 5.

Issue 6

The DESCRIPTION is updated to say that the referenced object is closed ``if this is the last reference to it''.

Issue 7

The *dlopen*( ) function is moved from the XSI option to Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0069 [74] is applied.

Issue 8

Austin Group Defect 993 is applied, adding *dladdr*( ) to the SEE ALSO section.

28005 **NAME**

28006 dlerror — get diagnostic information

28007 **SYNOPSIS**

28008
```
#include <dlfcn.h>
```

28009
```
char *dlerror(void);
```

28010 **DESCRIPTION**

28011 The *dlerror*( ) function shall return a null-terminated character string (with no trailing
28012 <newline>) that describes the last error that occurred during dynamic linking processing. If no
28013 dynamic linking errors have occurred since the last invocation of *dlerror*( ), *dlerror*( ) shall return
28014 NULL. Thus, invoking *dlerror*( ) a second time, immediately following a prior invocation, shall
28015 result in NULL being returned.

28016 It is implementation-defined whether or not the *dlerror*( ) function is thread-safe. A thread-safe
28017 implementation shall return only errors that occur on the current thread.

28018 **RETURN VALUE**

28019 If successful, *dlerror*( ) shall return a null-terminated character string; otherwise, NULL shall be
28020 returned.

28021 The application shall not modify the string returned. The returned pointer might be invalidated
28022 or the string content might be overwritten by a subsequent call to *dlerror*( ) in the same thread (if
28023 *dlerror*( ) is thread-safe) or in any thread (if *dlerror*( ) is not thread-safe). The returned pointer
28024 might also be invalidated if the calling thread is terminated.

28025 **ERRORS**

28026 No errors are defined.

28027 **EXAMPLES**

28028 The following example prints out the last dynamic linking error:

28029
```
...
#include <dlfcn.h>
```

28031
```
char *errstr;
```

28032
```
errstr = dlerror();
if (errstr != NULL)
    printf ("A dynamic linking error occurred: (%s)\n", errstr);
...
```

28036 **APPLICATION USAGE**

28037 Depending on the application environment with respect to asynchronous execution events, such
28038 as signals or other asynchronous computation sharing the address space, conforming
28039 applications should use a critical section to retrieve the error pointer and buffer.

28040 **RATIONALE**

28041 None.

28042 **FUTURE DIRECTIONS**

28043 None.

28044 **SEE ALSO**

28045 *dladdr*( ), *dlclose*( ), *dlopen*( ), *dlsym*( )

28046 XBD **<dlfcn.h>**

28047 **CHANGE HISTORY**

28048 First released in Issue 5.

28049 **Issue 6**

28050 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

28051 **Issue 7**

28052 Austin Group Interpretation 1003.1-2001 #156 is applied.

28053 The *dlerror*( ) function is moved from the XSI option to the Base.

28054 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0070 [75], XSH/TC1-2008/0071 [97],
28055 and XSH/TC1-2008/0072 [133] are applied.

28056 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0082 [656] is applied.

28057 **Issue 8**

28058 Austin Group Defect 993 is applied, adding *dladdr*( ) to the SEE ALSO section.

28059 **NAME**

28060 dlopen — open a symbol table handle

28061 **SYNOPSIS**

28062 ```
#include <dlfcn.h>
```

28063 ```
void *dlopen(const char *file, int mode);
```

28064 **DESCRIPTION**

28065 The *dlopen*( ) function shall make the symbols (function identifiers and data object identifiers) in
28066 the executable object file specified by *file* available to the calling program.

28067 The class of executable object files eligible for this operation and the manner of their
28068 construction are implementation-defined, though typically such files are shared libraries or
28069 programs.

28070 Implementations may permit the construction of embedded dependencies in executable object
28071 files. In such cases, a *dlopen*( ) operation shall load those dependencies in addition to the
28072 executable object file specified by *file*. Implementations may also impose specific constraints on
28073 the construction of programs that can employ *dlopen*( ) and its related services.

28074 A successful *dlopen*( ) shall return a symbol table handle which the caller may use on subsequent
28075 calls to *dlsym*( ) and *dlclose*( ).

28076 The value of this symbol table handle should not be interpreted in any way by the caller.

28077 The *file* argument is used to construct a pathname to the executable object file. If *file* contains a
28078 <slash> character, the *file* argument is used as the pathname for the file. Otherwise, *file* is used in
28079 an implementation-defined manner to yield a pathname.

28080 If *file* is a null pointer, *dlopen*( ) shall return a global symbol table handle for the currently
28081 running process image. This symbol table handle shall provide access to the symbols from an
28082 ordered set of executable object files consisting of the original program image file, any
28083 executable object files loaded at program start-up as specified by that process file (for example,
28084 shared libraries), and the set of executable object files loaded using *dlopen*( ) operations with the
28085 RTLD_GLOBAL flag. As the latter set of executable object files can change during execution, the
28086 set of symbols made available by this symbol table handle can also change dynamically.

28087 Only a single copy of an executable object file shall be brought into the address space, even if
28088 *dlopen*( ) is invoked multiple times in reference to the executable object file, and even if different
28089 pathnames are used to reference the executable object file.

28090 The *mode* parameter describes how *dlopen*( ) shall operate upon *file* with respect to the processing
28091 of relocations and the scope of visibility of the symbols provided within *file*. When an
28092 executable object file is brought into the address space of a process, it may contain references to
28093 symbols whose addresses are not known until the executable object file is loaded.

28094 These references shall be relocated before the symbols can be accessed. The *mode* parameter
28095 governs when these relocations take place and may have the following values:

28096 RTLD_LAZY Relocations shall be performed at an implementation-defined time, ranging
28097 from the time of the *dlopen*( ) call until the first reference to a given symbol
28098 occurs. Specifying RTLD_LAZY should improve performance on
28099 implementations supporting dynamic symbol binding since a process might
28100 not reference all of the symbols in an executable object file. And, for systems
28101 supporting dynamic symbol resolution for normal process execution, this
28102 behavior mimics the normal handling of process execution.

RTLD_NOW All necessary relocations shall be performed when the executable object file is first loaded. This may waste some processing if relocations are performed for symbols that are never referenced. This behavior may be useful for applications that need to know that all symbols referenced during execution will be available before *dlopen*( ) returns.

Any executable object file loaded by *dlopen*( ) that requires relocations against global symbols can reference the symbols in the original process image file, any executable object files loaded at program start-up, from the initial process image itself, from any other executable object file included in the same *dlopen*( ) invocation, and any executable object files that were loaded in any *dlopen*( ) invocation and which specified the RTLD_GLOBAL flag. To determine the scope of visibility for the symbols loaded with a *dlopen*( ) invocation, the *mode* parameter should be a bitwise-inclusive OR with one of the following values:

RTLD_GLOBAL The executable object file's symbols shall be made available for relocation processing of any other executable object file. In addition, symbol lookup using *dlopen*(NULL,*mode*) and an associated *dlsym*( ) allows executable object files loaded with this mode to be searched.

RTLD_LOCAL The executable object file's symbols shall not be made available for relocation processing of any other executable object file.

If neither RTLD_GLOBAL nor RTLD_LOCAL is specified, the default behavior is unspecified.

If an executable object file is specified in multiple *dlopen*( ) invocations, *mode* is interpreted at each invocation.

If RTLD_NOW has been specified, all relocations shall have been completed rendering further RTLD_NOW operations redundant and any further RTLD_LAZY operations irrelevant.

If RTLD_GLOBAL has been specified, the executable object file shall maintain the RTLD_GLOBAL status regardless of any previous or future specification of RTLD_LOCAL, as long as the executable object file remains in the address space (see *dlclose*( )). If there was a previous specification of RTLD_LOCAL, it is unspecified whether relocations after the new specification of RTLD_GLOBAL are made as if the previous specification had been RTLD_GLOBAL or as if the executable object file had not previously been loaded.

Symbols introduced into the process image through calls to *dlopen*( ) may be used in relocation activities. Symbols so introduced may duplicate symbols already defined by the program or previous *dlopen*( ) operations. To resolve the ambiguities such a situation might present, the resolution of a symbol reference to symbol definition is based on a symbol resolution order. Two such resolution orders are defined: load order and dependency order. Load order establishes an ordering among symbol definitions, such that the first definition loaded (including definitions from the process image file and any dependent executable object files loaded with it) has priority over executable object files added later (by *dlopen*( )). Load ordering is used in relocation processing. Dependency ordering uses a breadth-first order starting with a given executable object file, then all of its dependencies, then any dependents of those, iterating until all dependencies are satisfied. With the exception of the global symbol table handle obtained via a *dlopen*( ) operation with a null pointer as the *file* argument, dependency ordering is used by the *dlsym*( ) function. Load ordering is used in *dlsym*( ) operations upon the global symbol table handle.

When an executable object file is first made accessible via *dlopen*( ), it and its dependent executable object files are added in dependency order. Once all the executable object files are added, relocations are performed using load order. Note that if an executable object file or its dependencies had been previously loaded, the load and dependency orders may yield different

resolutions.

The symbols introduced by *dlopen*( ) operations and available through *dlsym*( ) are at a minimum those which are exported as identifiers of global scope by the executable object file. Typically, such identifiers shall be those that were specified in (for example) C source code as having **extern** linkage. The precise manner in which an implementation constructs the set of exported symbols for an executable object file is implementation-defined.

RETURN VALUE

Upon successful completion, *dlopen*( ) shall return a symbol table handle. If *file* cannot be found, cannot be opened for reading, is not of an appropriate executable object file format for processing by *dlopen*( ), or if an error occurs during the process of loading *file* or relocating its symbolic references, *dlopen*( ) shall return a null pointer. More detailed diagnostic information shall be available through *dlerror*( ).

ERRORS

No errors are defined.

EXAMPLES

Refer to *dlsym*( ).

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

dladdr( ), *dlclose*( ), *dlerror*( ), *dlsym*( )

XBD **<dlfcn.h>**

CHANGE HISTORY

First released in Issue 5.

Issue 6

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/21 is applied, changing the default behavior in the DESCRIPTION when neither RTLD_GLOBAL nor RTLD_LOCAL are specified from implementation-defined to unspecified.

Issue 7

The *dlopen*( ) function is moved from the XSI option to the Base.

The EXAMPLES section is updated to refer to *dlsym*( ).

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0073 [74] is applied.

Issue 8

Austin Group Defect 982 is applied, clarifying the requirements when changing from RTLD_LOCAL to RTLD_GLOBAL.

Austin Group Defect 993 is applied, adding *dladdr*( ) to the SEE ALSO section.

NAME

dlsym — get the address of a symbol from a symbol table handle

SYNOPSIS

```
#include <dlfcn.h>

void *dlsym(void *restrict handle, const char *restrict name);
```

DESCRIPTION

The *dlsym*( ) function shall obtain the address of a symbol (a function identifier or a data object identifier) defined in the symbol table identified by the *handle* argument. The *handle* argument is a symbol table handle returned from a call to *dlopen*( ) (and which has not since been released by a call to *dlclose*( )), and *name* is the symbol's name as a character string. The return value from *dlsym*( ), converted from type pointer to **void** to a pointer to the type of the named symbol, can be used to call (in the case of a function) or access the contents of (in the case of a data object) the named symbol.

The *dlsym*( ) function shall search for the named symbol in the symbol table referenced by *handle*. If the symbol table was created with lazy loading (see RTLD_LAZY in *dlopen*( )), load ordering shall be used in *dlsym*( ) operations to relocate executable object files needed to resolve the symbol. The symbol resolution algorithm used shall be dependency order as described in *dlopen*( ).

The RTLD_DEFAULT and RTLD_NEXT symbolic constants (which may be defined in **<dlfcn.h>**) are reserved for future use as special values that applications may be allowed to use for *handle*.

RETURN VALUE

Upon successful completion, if *name* names a function identifier, *dlsym*( ) shall return the address of the function converted from type pointer to function to type pointer to **void**; otherwise, *dlsym*( ) shall return the address of the data object associated with the data object identifier named by *name* converted from a pointer to the type of the data object to a pointer to **void**. If *handle* does not refer to a valid symbol table handle or if the symbol named by *name* cannot be found in the symbol table associated with *handle*, *dlsym*( ) shall return a null pointer.

More detailed diagnostic information shall be available through *dlerror*( ).

ERRORS

No errors are defined.

EXAMPLES

The following example shows how *dlopen*( ) and *dlsym*( ) can be used to access either a function or a data object. For simplicity, error checking has been omitted.

```
void *handle;
int (*fptr)(int), *iptr, result;
/* open the needed symbol table */
handle = dlopen("/usr/home/me/libfoo.so", RTLD_LOCAL | RTLD_LAZY);
/* find the address of the function my_function */
fptr = (int (*)(int))dlsym(handle, "my_function");
/* find the address of the data object my_object */
iptr = (int *)dlsym(handle, "my_OBJ");
/* invoke my_function, passing the value of my_OBJ as the parameter */
result = (*fptr)(*iptr);
```


APPLICATION USAGE

The following special purpose values for *handle* are reserved for future use and have the indicated meanings:

RTLD_DEFAULT The identifier lookup happens in the normal global scope; that is, a search for an identifier using *handle* would find the same definition as a direct use of this identifier in the program code.

RTLD_NEXT Specifies the next executable object file after this one that defines *name*. This one refers to the executable object file containing the invocation of *dlsym*( ). The next executable object file is the one found upon the application of a load order symbol resolution algorithm (see *dlopen*( )). The next symbol is either one of global scope (because it was introduced as part of the original process image or because it was added with a *dlopen*( ) operation including the RTLD_GLOBAL flag), or is in an executable object file that was included in the same *dlopen*( ) operation that loaded this one.

The RTLD_NEXT flag is useful to navigate an intentionally created hierarchy of multiply-defined symbols created through interposition. For example, if a program wished to create an implementation of *malloc*( ) that embedded some statistics gathering about memory allocations, such an implementation could use the real *malloc*( ) definition to perform the memory allocation — and itself only embed the necessary logic to implement the statistics gathering function.

Note that conversion from a **void** * pointer to a function pointer as in:

```
fptr = (int (*)(int))dlsym(handle, "my_function");
```

is not defined by the ISO C standard. This standard requires this conversion to work correctly on conforming implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

dladdr( ), *dlclose*( ), *dlerror*( ), *dlopen*( )

XBD **<dlfcn.h>**

CHANGE HISTORY

First released in Issue 5.

Issue 6

The **restrict** keyword is added to the *dlsym*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

The RTLD_DEFAULT and RTLD_NEXT flags are reserved for future use.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/14 is applied, correcting an example, and adding text to the RATIONALE describing issues related to conversion of pointers to functions and back again.

Issue 7

The *dlsym*( ) function is moved from the XSI option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0074 [74] is applied.

Issue 8

Austin Group Defect 993 is applied, adding *dladdr*( ) to the SEE ALSO section.

Austin Group Defect 1644 is applied, clarifying that the return value from *dlsym*( ) can be converted from type pointer to **void** to a pointer to the type of the named symbol using any valid conversion method, not just casting.

28280 **NAME**

28281 dngettext, dngettext_l — message handling functions

28282 **SYNOPSIS**

28283 ```
#include <libintl.h>
```

28284 ```
char *dngettext(const char *domainname, const char *msgid,
28285     const char *msgid_plural, unsigned long int n);
28286 char *dngettext_l(const char *domainname, const char *msgid,
28287     const char *msgid_plural, unsigned long int n,
28288     locale_t locale);
```

28289 **DESCRIPTION**

28290 Refer to *gettext*.

28291 **NAME**

28292 dprintf — print formatted output

28293 **SYNOPSIS**

28294 CX
```
#include <stdio.h>

int dprintf(int fildes, const char *restrict format, ...);
```

28296 **DESCRIPTION**

28297 Refer to *fprintf( )*.

NAME

drand48, erand48, jrand48, lcong48, lrand48, mrand48, nrand48, seed48, srand48 — generate uniformly distributed pseudo-random numbers

SYNOPSIS

XSI

```
#include <stdlib.h>

double drand48(void);
double erand48(unsigned short xsubi[3]);
long jrand48(unsigned short xsubi[3]);
void lcong48(unsigned short param[7]);
long lrand48(void);
long mrand48(void);
long nrand48(unsigned short xsubi[3]);
unsigned short *seed48(unsigned short seed16v[3]);
void srand48(long seedval);
```

DESCRIPTION

This family of functions shall generate pseudo-random numbers using a linear congruential algorithm and 48-bit integer arithmetic.

The *drand48*( ) and *erand48*( ) functions shall return non-negative, double-precision, floating-point values, uniformly distributed over the interval [0.0,1.0).

The *lrand48*( ) and *nrand48*( ) functions shall return non-negative, long integers, uniformly distributed over the interval $[0,2^{31})$.

The *mrand48*( ) and *jrand48*( ) functions shall return signed long integers uniformly distributed over the interval $[-2^{31},2^{31})$.

The *srand48*( ), *seed48*( ), and *lcong48*( ) functions are initialization entry points, one of which should be invoked before either *drand48*( ), *lrand48*( ), or *mrand48*( ) is called. (Although it is not recommended practice, constant default initializer values shall be supplied automatically if *drand48*( ), *lrand48*( ), or *mrand48*( ) is called without a prior call to an initialization entry point.) The *erand48*( ), *nrand48*( ), and *jrand48*( ) functions do not require an initialization entry point to be called first.

All the routines work by generating a sequence of 48-bit integer values, X_i, according to the linear congruential formula:

$$X_{n+1} = (aX_n + c)_{\mathrm{mod}\ m} \qquad n \geq 0$$

The parameter $m = 2^{48}$; hence 48-bit integer arithmetic is performed. Unless *lcong48*( ) is invoked, the multiplier value a and the addend value c are given by:

$$a = 5\mathrm{DEECE66D}_{16} = 273673163155_8$$

$$c = \mathrm{B}_{16} = 13_8$$

The value returned by any of the *drand48*( ), *erand48*( ), *jrand48*( ), *lrand48*( ), *mrand48*( ), or *nrand48*( ) functions is computed by first generating the next 48-bit X_i in the sequence. X_i is then converted to the return value as follows:

- For *drand48*( ) and *erand48*( ) the value shall be 2^{-48} times X_i.
- For *jrand48*( ) and *mrand48*( ) the value shall be the largest integer not greater than 2^{-16} times X_i.

- For *lrand48*( ) and *nrand48*( ) the value shall be the largest integer not greater than 2^{-17} times X_i.

The *drand48*( ), *lrand48*( ), and *mrand48*( ) functions store the last 48-bit X_i generated in an internal buffer; that is why the application shall ensure that these are initialized prior to being invoked. The *erand48*( ), *nrand48*( ), and *jrand48*( ) functions require the calling program to provide storage for the successive X_i values in the array specified as an argument when the functions are invoked. That is why these routines do not have to be initialized; the calling program merely has to place the desired initial value of X_i into the array and pass it as an argument. By using different arguments, *erand48*( ), *nrand48*( ), and *jrand48*( ) allow separate modules of a large program to generate several *independent* streams of pseudo-random numbers; that is, the sequence of numbers in each stream shall *not* depend upon how many times the routines are called to generate numbers for the other streams.

The initializer function *srand48*( ) sets the high-order 32 bits of X_i to the low-order 32 bits contained in its argument. The low-order 16 bits of X_i are set to the arbitrary value $330E_{16}$.

The initializer function *seed48*( ) sets the value of X_i to the 48-bit value specified in the argument array. The low-order 16 bits of X_i are set to the low-order 16 bits of *seed16v*[*0*]. The mid-order 16 bits of X_i are set to the low-order 16 bits of *seed16v*[*1*]. The high-order 16 bits of X_i are set to the low-order 16 bits of *seed16v*[*2*]. In addition, the previous value of X_i is copied into a 48-bit internal buffer, used only by *seed48*( ), and a pointer to this buffer is the value returned by *seed48*( ). This returned pointer, which can just be ignored if not needed, is useful if a program is to be restarted from a given point at some future time—use the pointer to get at and store the last X_i value, and then use this value to reinitialize via *seed48*( ) when the program is restarted.

The initializer function *lcong48*( ) allows the user to specify the initial X_i, the multiplier value *a*, and the addend value *c*. Argument array elements *param*[*0-2*] specify X_i, *param*[*3-5*] specify the multiplier *a*, and *param*[*6*] specifies the 16-bit addend *c*. After *lcong48*( ) is called, a subsequent call to either *srand48*( ) or *seed48*( ) shall restore the standard multiplier and addend values, *a* and *c*, specified above.

The *drand48*( ), *lrand48*( ), and *mrand48*( ) functions need not be thread-safe.

RETURN VALUE

As described in the DESCRIPTION above.

ERRORS

No errors are defined.

EXAMPLES

The following program tests that the required pseudo-random number generator is used by these functions.

```
#include <assert.h>
#include <stdlib.h>

int main() {
  {
    unsigned short xsubi[3] = {37174, 64810, 11603};
    double d = erand48(xsubi);
    assert(d >= 0.896);
    assert(d <= 0.897);
    assert(xsubi[0] == 22537);
    assert(xsubi[1] == 47966);
    assert(xsubi[2] == 58735);
    d = erand48(xsubi);
```



```
28387             assert(d >= 0.337);
28388             assert(d <= 0.338);
28389             assert(xsubi[0] == 37344);
28390             assert(xsubi[1] == 32911);
28391             assert(xsubi[2] == 22119);
28392             d = erand48(xsubi);
28393             assert(d >= 0.647);
28394             assert(d <= 0.648);
28395             assert(xsubi[0] == 23659);
28396             assert(xsubi[1] == 29872);
28397             assert(xsubi[2] == 42445);
28398             d = erand48(xsubi);
28399             assert(d >= 0.500);
28400             assert(d <= 0.501);
28401             assert(xsubi[0] == 31642);
28402             assert(xsubi[1] == 7875);
28403             assert(xsubi[2] == 32802);
28404             d = erand48(xsubi);
28405             assert(d >= 0.506);
28406             assert(d <= 0.507);
28407             assert(xsubi[0] == 64669);
28408             assert(xsubi[1] == 14399);
28409             assert(xsubi[2] == 33170);
28410           }
28411           {
28412             unsigned short xsubi[3] = {25175, 11052, 45015};
28413             assert(jrand48(xsubi) == 1699503220);
28414             assert(xsubi[0] == 2326);
28415             assert(xsubi[1] == 23668);
28416             assert(xsubi[2] == 25932);
28417             assert(jrand48(xsubi) == -992276007);
28418             assert(xsubi[0] == 41577);
28419             assert(xsubi[1] == 4569);
28420             assert(xsubi[2] == 50395);
28421             assert(jrand48(xsubi) == -19535776);
28422             assert(xsubi[0] == 31936);
28423             assert(xsubi[1] == 59488);
28424             assert(xsubi[2] == 65237);
28425             assert(jrand48(xsubi) == 79438377);
28426             assert(xsubi[0] == 40395);
28427             assert(xsubi[1] == 8745);
28428             assert(xsubi[2] == 1212);
28429             assert(jrand48(xsubi) == -1258917728);
28430             assert(xsubi[0] == 37242);
28431             assert(xsubi[1] == 28832);
28432             assert(xsubi[2] == 46326);
28433           }
28434           {
28435             unsigned short xsubi[3] = {546, 33817, 23389};
28436             assert(nrand48(xsubi) == 914920692);
28437             assert(xsubi[0] == 29829);
```



```
            assert(xsubi[1] == 10728);
            assert(xsubi[2] == 27921);
            assert(nrand48(xsubi) == 754104482);
            assert(xsubi[0] == 6828);
            assert(xsubi[1] == 28997);
            assert(xsubi[2] == 23013);
            assert(nrand48(xsubi) == 609453945);
            assert(xsubi[0] == 58183);
            assert(xsubi[1] == 3826);
            assert(xsubi[2] == 18599);
            assert(nrand48(xsubi) == 1878644360);
            assert(xsubi[0] == 36678);
            assert(xsubi[1] == 44304);
            assert(xsubi[2] == 57331);
            assert(nrand48(xsubi) == 2114923686);
            assert(xsubi[0] == 58585);
            assert(xsubi[1] == 22861);
            assert(xsubi[2] == 64542);
        }
    }
```

APPLICATION USAGE

These functions should be avoided whenever non-trivial requirements (including safety) have to be fulfilled, unless seeded using *getentropy*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getentropy( ), *initstate*( ), *rand*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

A note indicating that the *drand48*( ), *lrand48*( ), and *mrand48*( ) functions need not be reentrant is added to the DESCRIPTION.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0083 [743] is applied.

Issue 8

Austin Group Defect 1107 is applied, clarifying how the return value is calculated from X_i for each function.

Austin Group Defect 1134 is applied, adding *getentropy*( ).

NAME

dup, dup2, dup3 — duplicate an open file descriptor

SYNOPSIS

```
#include <unistd.h>

int dup(int fildes);
int dup2(int fildes, int fildes2);
int dup3(int fildes, int fildes2, int flag);
```

DESCRIPTION

The *dup*( ) function provides an alternative interface to the service provided by *fcntl*( ) using the F_DUPFD command. The call *dup*(*fildes*) shall be equivalent to:

```
fcntl(fildes, F_DUPFD, 0);
```

The *dup2*( ) function shall cause the file descriptor *fildes2* to refer to the same open file description as the file descriptor *fildes* and to share any locks, and shall return *fildes2*. If *fildes2* is already a valid open file descriptor, it shall be closed first, unless *fildes* is equal to *fildes2* in which case *dup2*( ) shall return *fildes2* without closing it. If the close operation fails to close *fildes2*, *dup2*( ) shall return –1 without changing the open file description to which *fildes2* refers. If *fildes* is not a valid file descriptor, *dup2*( ) shall return –1 and shall not close *fildes2*. If *fildes2* is less than 0 or greater than or equal to {OPEN_MAX}, *dup2*( ) shall return –1 with *errno* set to [EBADF].

Upon successful completion, if *fildes* is not equal to *fildes2*, the FD_CLOEXEC and FD_CLOFORK flags associated with *fildes2* shall be cleared. If *fildes* is equal to *fildes2*, the FD_CLOEXEC and FD_CLOFORK flags associated with *fildes2* shall not be changed.

The *dup3*( ) function shall be equivalent to the *dup2*( ) function, except that it shall be an error if *fildes* is equal to *fildes2*, and the state of FD_CLOEXEC and FD_CLOFORK on the *fildes2* file descriptor shall be determined solely by the *flag* argument, which can be constructed from a bitwise-inclusive OR of flags from the following list:

O_CLOEXEC Atomically set the FD_CLOEXEC flag on *fildes2*.

O_CLOFORK Atomically set the FD_CLOFORK flag on *fildes2*.

TYM If *fildes* refers to a typed memory object, the result of the *dup2*( ) or *dup3*( ) functions is unspecified.

RETURN VALUE

Upon successful completion a non-negative integer, namely the file descriptor, shall be returned; otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *dup*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid open file descriptor.

[EMFILE] All file descriptors available to the process are currently open.

The *dup2*( ) and *dup3*( ) functions shall fail if:

[EBADF] The *fildes* argument is not a valid open file descriptor or the argument *fildes2* is negative or greater than or equal to {OPEN_MAX}.

[EINTR] The function was interrupted by a signal.

The *dup3*( ) function shall fail if:

28523 [EINVAL] The *fildes* and *fildes2* arguments are equal.

28524 The *dup2*( ) and *dup3*( ) functions may fail if:

28525 [EIO] An I/O error occurred while attempting to close *fildes2*.

28526 The *dup3*( ) function may fail if:

28527 [EINVAL] The value of the *flag* argument is invalid.

28528 **EXAMPLES**

28529 **Redirecting Standard Output to a File**

28530 The following example closes standard output for the current processes, re-assigns standard
28531 output to go to the file referenced by *pfd*, and closes the original file descriptor to clean up.

```
28532 #include <unistd.h>
28533 ...
28534 int pfd;
28535 ...
28536 close(1);
28537 dup(pfd);
28538 close(pfd);
28539 ...
```

28540 **Redirecting Error Messages**

28541 The following example redirects messages from *stderr* to *stdout*.

```
28542 #include <unistd.h>
28543 ...
28544 dup2(1, 2);
28545 ...
```

28546 **APPLICATION USAGE**

28547 Implementations may use file descriptors that must be inherited into child processes for the
28548 child process to remain conforming, such as for message catalog or tracing purposes. Therefore,
28549 an application that calls *dup2*( ) with an arbitrary integer for *fildes2* risks non-conforming
28550 behavior, and *dup2*( ) can only portably be used to overwrite file descriptor values that the
28551 application has obtained through explicit actions, or for the three file descriptors corresponding
28552 to the standard file streams. In order to avoid a race condition of leaking an unintended file
28553 descriptor into a child process or executed program, an application should consider opening all
28554 file descriptors with the FD_CLOFORK or FD_CLOEXEC flag, or both flags, set unless the file
28555 descriptor is intended to be inherited by child processes or executed programs, respectively.

28556 **RATIONALE**

28557 The *dup*( ) function is redundant. Its services are also provided by the *fcntl*( ) function. It has been
28558 included in this volume of POSIX.1-2024 primarily for historical reasons, since many existing
28559 applications use it. On the other hand, the *dup2*( ) function provides unique services, as no other
28560 interface is able to atomically replace an existing file descriptor.

28561 The *dup2*( ) function is not marked obsolescent because it presents a type-safe version of
28562 functionality provided in a type-unsafe version by *fcntl*( ). It is used in the POSIX Ada binding.

28563 The *dup2*( ) function is not intended for use in critical regions as a synchronization mechanism.

28564 In the description of [EBADF], the case of *fildes* being out of range is covered by the given case of

fildes not being valid. The descriptions for *fildes* and *fildes2* are different because the only kind of invalidity that is relevant for *fildes2* is whether it is out of range; that is, it does not matter whether *fildes2* refers to an open file when the *dup2*( ) call is made.

The *dup3*( ) function with the O_CLOEXEC and O_CLOFORK flags is necessary to avoid a data race in multi-threaded applications. Without O_CLOFORK, a file descriptor is leaked into a child process created by one thread in the window between another thread creating a file descriptor with *dup2*( ) and then using *fcntl*( ) to set the FD_CLOFORK flag. Without O_CLOEXEC, a file descriptor intentionally inherited by child processes is similarly leaked into an executed program if FD_CLOEXEC is not set atomically. The safe counterpart for avoiding the same race with *dup*( ) is the use of the F_DUPFD_CLOFORK or F_DUPFD_CLOEXEC action of the *fcntl*( ) function.

FUTURE DIRECTIONS

None.

SEE ALSO

close( ), *fcntl*( ), *open*( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7

SD5-XSH-ERN-187 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0075 [149,428] and XSH/TC1-2008/0076 [149] are applied.

Issue 8

Austin Group Defects 411, 1318, 1483, and 1577 are applied, adding *dup3*( ) and FD_CLOFORK.

NAME

duplocale — duplicate a locale object

SYNOPSIS

CX
```
#include <locale.h>

locale_t duplocale(locale_t locobj);
```

DESCRIPTION

The *duplocale*( ) function shall create a duplicate copy of the locale object referenced by the *locobj* argument.

If the *locobj* argument is LC_GLOBAL_LOCALE, *duplocale*( ) shall create a new locale object containing a copy of the global locale determined by the *setlocale*( ) function.

The behavior is undefined if the *locobj* argument is not a valid locale object handle.

RETURN VALUE

Upon successful completion, the *duplocale*( ) function shall return a handle for a new locale object. Otherwise, *duplocale*( ) shall return (**locale_t**)0 and set *errno* to indicate the error.

ERRORS

The *duplocale*( ) function shall fail if:

[ENOMEM] There is not enough memory available to create the locale object or load the locale data.

EXAMPLES

Constructing an Altered Version of an Existing Locale Object

The following example shows a code fragment to create a slightly altered version of an existing locale object. The function takes a locale object and a locale name and it replaces the *LC_TIME* category data in the locale object with that from the named locale.

```
#include <locale.h>
...

locale_t
with_changed_lc_time (locale_t obj, const char *name)
{
    locale_t retval = duplocale (obj);
    if (retval != (locale_t) 0)
    {
        locale_t changed = newlocale (LC_TIME_MASK, name, retval);
        if (changed == (locale_t) 0)
            /* An error occurred. Free all allocated resources. */
            freelocale (retval);
        retval = changed;
    }
    return retval;
}
```


APPLICATION USAGE

The use of the *duplocale*( ) function is recommended for situations where a locale object is being used in multiple places, and it is possible that the lifetime of the locale object might end before all uses are finished. Another reason to duplicate a locale object is if a slightly modified form is needed. This can be achieved by a call to *newlocale*( ) following the *duplocale*( ) call.

As with the *newlocale*( ) function, handles for locale objects created by the *duplocale*( ) function should be released by a corresponding call to *freelocale*( ).

The *duplocale*( ) function can also be used in conjunction with *uselocale*((**locale_t**)0). This returns the locale in effect for the calling thread, but can have the value LC_GLOBAL_LOCALE. Passing LC_GLOBAL_LOCALE to functions such as *isalnum_l*( ) results in undefined behavior, but applications can convert it into a usable locale object by using *duplocale*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

freelocale( ), *newlocale*( ), *uselocale*( )

XBD **<locale.h>**

CHANGE HISTORY

First released in Issue 7.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0077 [283,301], XSH/TC1-2008/0078 [283], and XSH/TC1-2008/0079 [301] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0084 [753] is applied.

NAME

encrypt — encoding function (**CRYPT**)

SYNOPSIS

OB XSI
```
#include <unistd.h>

void encrypt(char block[64], int edflag);
```

DESCRIPTION

The *encrypt*( ) function shall provide access to an implementation-defined encoding algorithm. The key generated by *setkey*( ) is used to encrypt the string *block* with *encrypt*( ).

The *block* argument to *encrypt*( ) shall be an array of length 64 bytes containing only the bytes with values of 0 and 1. The array is modified in place to a similar array using the key set by *setkey*( ). If *edflag* is 0, the argument is encoded. If *edflag* is 1, the argument may be decoded (see the APPLICATION USAGE section); if the argument is not decoded, *errno* shall be set to [ENOSYS].

The *encrypt*( ) function shall not change the setting of *errno* if successful. An application wishing to check for error situations should set *errno* to 0 before calling *encrypt*( ). If *errno* is non-zero on return, an error has occurred.

The *encrypt*( ) function need not be thread-safe.

RETURN VALUE

The *encrypt*( ) function shall not return a value.

ERRORS

The *encrypt*( ) function shall fail if:

[ENOSYS] The functionality is not supported on this implementation.

EXAMPLES

None.

APPLICATION USAGE

The *encrypt*( ) function historically used the DES block cipher, which is no longer considered secure.

In some environments, decoding might not be implemented. This is related to some Government restrictions on encryption and decryption routines. Historical practice has been to ship a different version of the encryption library without the decryption feature in the routines supplied. Thus the exported version of *encrypt*( ) does encoding but not decoding.

RATIONALE

None.

FUTURE DIRECTIONS

The *encrypt*( ) function may be removed in a future version.

SEE ALSO

crypt( ), *setkey*( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

28691 **Issue 5**

28692 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

28693 **Issue 6**

28694 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

28695 **Issue 7**

28696 Austin Group Interpretation 1003.1-2001 #156 is applied.

28697 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0085 [899] is applied.

28698 **Issue 8**

28699 Austin Group Defect 1192 is applied, marking the *encrypt*( ) function as obsolescent.

28700 **NAME**

28701 endgrent, getgrent, setgrent — group database entry functions

28702 **SYNOPSIS**

28703 XSI
```
#include <grp.h>

void endgrent(void);
struct group *getgrent(void);
void setgrent(void);
```

28707 **DESCRIPTION**

28708 The *getgrent*( ) function shall return a pointer to a structure containing the broken-out fields of an
28709 entry in the group database. If the group database is not already open, *getgrent*( ) shall open it
28710 and return a pointer to a **group** structure containing the first entry in the database. Thereafter, it
28711 shall return a pointer to a **group** structure containing the next **group** structure in the group
28712 database, so successive calls may be used to search the entire database.

28713 An implementation that provides extended security controls may impose further
28714 implementation-defined restrictions on accessing the group database. In particular, the system
28715 may deny the existence of some or all of the group database entries associated with groups other
28716 than those groups associated with the caller and may omit users other than the caller from the
28717 list of members of groups in database entries that are returned.

28718 The *setgrent*( ) function shall rewind the group database so that the next *getgrent*( ) call returns
28719 the first entry, allowing repeated searches.

28720 The *endgrent*( ) function shall close the group database.

28721 The *setgrent*( ) and *endgrent*( ) functions shall not change the setting of *errno* if successful.

28722 On error, the *setgrent*( ) and *endgrent*( ) functions shall set *errno* to indicate the error.

28723 Since no value is returned by the *setgrent*( ) and *endgrent*( ) functions, an application wishing to
28724 check for error situations should set *errno* to 0, then call the function, then check *errno*.

28725 These functions need not be thread-safe.

28726 **RETURN VALUE**

28727 On successful completion, *getgrent*( ) shall return a pointer to a **group** structure. On end-of-file,
28728 *getgrent*( ) shall return a null pointer and shall not change the setting of *errno*. On error,
28729 *getgrent*( ) shall return a null pointer and *errno* shall be set to indicate the error.

28730 The application shall not modify the structure to which the return value points, nor any storage
28731 areas pointed to by pointers within the structure. The returned pointer, and pointers within the
28732 structure, might be invalidated or the structure or the storage areas might be overwritten by a
28733 subsequent call to *getgrgid*( ), *getgrnam*( ), or *getgrent*( ). The returned pointer, and pointers
28734 within the structure, might also be invalidated if the calling thread is terminated.

28735 **ERRORS**

28736 These functions may fail if:

28737 [EINTR] A signal was caught during the operation.

28738 [EIO] An I/O error has occurred.

28739 In addition, the *getgrent*( ) and *setgrent*( ) functions may fail if:

28740 [EMFILE] All file descriptors available to the process are currently open.

28741 [ENFILE] The maximum allowable number of files is currently open in the system.

28742 **EXAMPLES**

28743 None.

28744 **APPLICATION USAGE**

28745 These functions are provided due to their historical usage. Applications should avoid
28746 dependencies on fields in the group database, whether the database is a single file, or where in
28747 the file system name space the database resides. Applications should use *getgrnam*( ) and
28748 *getgrgid*( ) whenever possible because it avoids these dependencies.

28749 **RATIONALE**

28750 None.

28751 **FUTURE DIRECTIONS**

28752 None.

28753 **SEE ALSO**

28754 *endpwent*( ), *getgrgid*( ), *getgrnam*( ), *getlogin*( )

28755 XBD **<grp.h>**

28756 **CHANGE HISTORY**

28757 First released in Issue 4, Version 2.

28758 **Issue 5**

28759 Moved from X/OPEN UNIX extension to BASE.

28760 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
28761 VALUE section.

28762 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

28763 **Issue 6**

28764 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

28765 **Issue 7**

28766 Austin Group Interpretation 1003.1-2001 #156 is applied.

28767 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

28768 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0080 [75] is applied.

28769 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0086 [493], XSH/TC2-2008/0087 [656],
28770 and XSH/TC2-2008/0088 [493] are applied.

28771 **NAME**

28772 endhostent, gethostent, sethostent — network host database functions

28773 **SYNOPSIS**

28774
```
#include <netdb.h>
```

28775
```
void endhostent(void);
```
28776
```
struct hostent *gethostent(void);
```
28777
```
void sethostent(int stayopen);
```

28778 **DESCRIPTION**

28779 These functions shall retrieve information about hosts. This information is considered to be
28780 stored in a database that can be accessed sequentially or randomly. The implementation of this
28781 database is unspecified.

28782 **Note:** In many cases this database is implemented by the Domain Name System, as documented in
28783 RFC 1034, RFC 1035, and RFC 3596.

28784 The *sethostent*( ) function shall open a connection to the database and set the next entry for
28785 retrieval to the first entry in the database. If the *stayopen* argument is non-zero, the connection
28786 shall not be closed by a call to *gethostent*( ), and the implementation may maintain an open file
28787 descriptor. If a file descriptor is opened, the FD_CLOEXEC flag shall be set; see **<fcntl.h>**.

28788 The *gethostent*( ) function shall read the next entry in the database, opening and closing a
28789 connection to the database as necessary.

28790 Entries shall be returned in **hostent** structures.

28791 The *endhostent*( ) function shall close the connection to the database, releasing any open file
28792 descriptor.

28793 These functions need not be thread-safe.

28794 **RETURN VALUE**

28795 Upon successful completion, the *gethostent*( ) function shall return a pointer to a **hostent**
28796 structure if the requested entry was found, and a null pointer if the end of the database was
28797 reached or the requested entry was not found.

28798 The application shall not modify the structure to which the return value points, nor any storage
28799 areas pointed to by pointers within the structure. The returned pointer, and pointers within the
28800 structure, might be invalidated or the structure or the storage areas might be overwritten by a
28801 subsequent call to *gethostent*( ). The returned pointer, and pointers within the structure, might
28802 also be invalidated if the calling thread is terminated.

28803 **ERRORS**

28804 The *gethostent*( ) and *sethostent*( ) functions may fail if:

28805 [EMFILE] All file descriptors available to the process are currently open.

28806 [ENFILE] The maximum allowable number of files is currently open in the system.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endservent( )

XBD **<fcntl.h>**, **<netdb.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0081 [75,428] and XSH/TC1-2008/0082 [75] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0089 [656] is applied.

Issue 8

Austin Group Defect 368 is applied, adding a requirement for FD_CLOEXEC to be set if a file descriptor is opened, and adding the [EMFILE] and [ENFILE] errors.

Austin Group Defect 1685 is applied, updating RFC references.

NAME

endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent — network database functions

SYNOPSIS

```
#include <netdb.h>

void endnetent(void);
struct netent *getnetbyaddr(uint32_t net, int type);
struct netent *getnetbyname(const char *name);
struct netent *getnetent(void);
void setnetent(int stayopen);
```

DESCRIPTION

These functions shall retrieve information about networks. This information is considered to be stored in a database that can be accessed sequentially or randomly. The implementation of this database is unspecified.

The *setnetent*( ) function shall open and rewind the database. If the *stayopen* argument is non-zero, the connection to the *net* database shall not be closed after each call to *getnetent*( ) (either directly, or indirectly through one of the other *getnet**( ) functions), and the implementation may maintain an open file descriptor to the database. If a file descriptor is used, the FD_CLOEXEC flag shall be set; see **<fcntl.h>**.

The *getnetent*( ) function shall read the next entry of the database, opening and closing a connection to the database as necessary.

The *getnetbyaddr*( ) function shall search the database from the beginning, and find the first entry for which the address family specified by *type* matches the *n_addrtype* member and the network number *net* matches the *n_net* member, opening and closing a connection to the database as necessary. The *net* argument shall be the network number in host byte order.

The *getnetbyname*( ) function shall search the database from the beginning and find the first entry for which the network name specified by *name* matches the *n_name* member, opening and closing a connection to the database as necessary.

The *getnetbyaddr*( ), *getnetbyname*( ), and *getnetent*( ) functions shall each return a pointer to a **netent** structure, the members of which shall contain the fields of an entry in the network database.

The *endnetent*( ) function shall close the database, releasing any open file descriptor.

These functions need not be thread-safe.

RETURN VALUE

Upon successful completion, *getnetbyaddr*( ), *getnetbyname*( ), and *getnetent*( ) shall return a pointer to a **netent** structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer shall be returned.

The application shall not modify the structure to which the return value points, nor any storage areas pointed to by pointers within the structure. The returned pointer, and pointers within the structure, might be invalidated or the structure or the storage areas might be overwritten by a subsequent call to *getnetbyaddr*( ), *getnetbyname*( ), or *getnetent*( ). The returned pointer, and pointers within the structure, might also be invalidated if the calling thread is terminated.

28871 **ERRORS**

28872 The *getnetbyaddr*( ), *getnetbyname*( ), *getnetent*( ), and *setnetent*( ) functions may fail if:

28873 [EMFILE] All file descriptors available to the process are currently open.

28874 [ENFILE] The maximum allowable number of files is currently open in the system.

28875 **EXAMPLES**

28876 None.

28877 **APPLICATION USAGE**

28878 None.

28879 **RATIONALE**

28880 None.

28881 **FUTURE DIRECTIONS**

28882 None.

28883 **SEE ALSO**

28884 XBD **<fcntl.h>**, **<netdb.h>**

28885 **CHANGE HISTORY**

28886 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

28887 **Issue 7**

28888 Austin Group Interpretation 1003.1-2001 #156 is applied.

28889 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0083 [75] and XSH/TC1-2008/0084
28890 [75] are applied.

28891 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0090 [656] is applied.

28892 **Issue 8**

28893 Austin Group Defect 368 is applied, adding a requirement for FD_CLOEXEC to be set if a file
28894 descriptor is used, and adding the [EMFILE] and [ENFILE] errors.

NAME

endprotoent, getprotobyname, getprotobynumber, getprotoent, setprotoent — network protocol database functions

SYNOPSIS

```
#include <netdb.h>

void endprotoent(void);
struct protoent *getprotobyname(const char *name);
struct protoent *getprotobynumber(int proto);
struct protoent *getprotoent(void);
void setprotoent(int stayopen);
```

DESCRIPTION

These functions shall retrieve information about protocols. This information is considered to be stored in a database that can be accessed sequentially or randomly. The implementation of this database is unspecified.

The *setprotoent*( ) function shall open a connection to the database, and set the next entry to the first entry. If the *stayopen* argument is non-zero, the connection to the network protocol database shall not be closed after each call to *getprotoent*( ) (either directly, or indirectly through one of the other *getproto**( ) functions), and the implementation may maintain an open file descriptor for the database. If a file descriptor is used, the FD_CLOEXEC flag shall be set; see **<fcntl.h>**.

The *getprotobyname*( ) function shall search the database from the beginning and find the first entry for which the protocol name specified by *name* matches the *p_name* member, opening and closing a connection to the database as necessary.

The *getprotobynumber*( ) function shall search the database from the beginning and find the first entry for which the protocol number specified by *proto* matches the *p_proto* member, opening and closing a connection to the database as necessary.

The *getprotoent*( ) function shall read the next entry of the database, opening and closing a connection to the database as necessary.

The *getprotobyname*( ), *getprotobynumber*( ), and *getprotoent*( ) functions shall each return a pointer to a **protoent** structure, the members of which shall contain the fields of an entry in the network protocol database.

The *endprotoent*( ) function shall close the connection to the database, releasing any open file descriptor.

These functions need not be thread-safe.

RETURN VALUE

Upon successful completion, *getprotobyname*( ), *getprotobynumber*( ), and *getprotoent*( ) return a pointer to a **protoent** structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer is returned.

The application shall not modify the structure to which the return value points, nor any storage areas pointed to by pointers within the structure. The returned pointer, and pointers within the structure, might be invalidated or the structure or the storage areas might be overwritten by a subsequent call to *getprotobyname*( ), *getprotobynumber*( ), or *getprotoent*( ). The returned pointer, and pointers within the structure, might also be invalidated if the calling thread is terminated.

28938 **ERRORS**

28939 The *getprotobyname*( ), *getprotobynumber*( ), *getprotoent*( ), and *setprotoent*( ) functions may fail if:

28940 [EMFILE] All file descriptors available to the process are currently open.

28941 [ENFILE] The maximum allowable number of files is currently open in the system.

28942 **EXAMPLES**

28943 None.

28944 **APPLICATION USAGE**

28945 None.

28946 **RATIONALE**

28947 None.

28948 **FUTURE DIRECTIONS**

28949 None.

28950 **SEE ALSO**

28951 XBD **<fcntl.h>**, **<netdb.h>**

28952 **CHANGE HISTORY**

28953 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

28954 **Issue 7**

28955 Austin Group Interpretation 1003.1-2001 #156 is applied.

28956 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0085 [75] and XSH/TC1-2008/0086
28957 [75] are applied.

28958 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0091 [656] is applied.

28959 **Issue 8**

28960 Austin Group Defect 368 is applied, adding a requirement for FD_CLOEXEC to be set if a file
28961 descriptor is used, and adding the [EMFILE] and [ENFILE] errors.

NAME

endpwent, getpwent, setpwent — user database functions

SYNOPSIS

XSI
```
#include <pwd.h>

void endpwent(void);
struct passwd *getpwent(void);
void setpwent(void);
```

DESCRIPTION

These functions shall retrieve information about users.

The *getpwent*( ) function shall return a pointer to a structure containing the broken-out fields of an entry in the user database. Each entry in the user database contains a **passwd** structure. If the user database is not already open, *getpwent*( ) shall open it and return a pointer to a **passwd** structure containing the first entry in the database. Thereafter, it shall return a pointer to a **passwd** structure containing the next entry in the user database. Successive calls can be used to search the entire user database.

If an end-of-file or an error is encountered on reading, *getpwent*( ) shall return a null pointer.

An implementation that provides extended security controls may impose further implementation-defined restrictions on accessing the user database. In particular, the system may deny the existence of some or all of the user database entries associated with users other than the caller.

The *setpwent*( ) function shall rewind the user database so that the next *getpwent*( ) call returns the first entry, allowing repeated searches.

The *endpwent*( ) function shall close the user database.

The *setpwent*( ) and *endpwent*( ) functions shall not change the setting of *errno* if successful.

On error, the *setpwent*( ) and *endpwent*( ) functions shall set *errno* to indicate the error.

Since no value is returned by the *setpwent*( ) and *endpwent*( ) functions, an application wishing to check for error situations should set *errno* to 0, then call the function, then check *errno*.

These functions need not be thread-safe.

RETURN VALUE

On successful completion, *getpwent*( ) shall return a pointer to a **passwd** structure. On end-of-file, *getpwent*( ) shall return a null pointer and shall not change the setting of *errno*. On error, *getpwent*( ) shall return a null pointer and *errno* shall be set to indicate the error.

The application shall not modify the structure to which the return value points, nor any storage areas pointed to by pointers within the structure. The returned pointer, and pointers within the structure, might be invalidated or the structure or the storage areas might be overwritten by a subsequent call to *getpwuid*( ), *getpwnam*( ), or *getpwent*( ). The returned pointer, and pointers within the structure, might also be invalidated if the calling thread is terminated.

ERRORS

These functions may fail if:

[EINTR] A signal was caught during the operation.

[EIO] An I/O error has occurred.

In addition, *getpwent*( ) and *setpwent*( ) may fail if:

29004 [EMFILE] All file descriptors available to the process are currently open.

29005 [ENFILE] The maximum allowable number of files is currently open in the system.

29006 **EXAMPLES**

29007 **Searching the User Database**

29008 The following example uses the *getpwent*( ) function to get successive entries in the user
29009 database, returning a pointer to a **passwd** structure that contains information about each user.
29010 The call to *endpwent*( ) closes the user database and cleans up.

```
29011 #include <pwd.h>
29012 #include <stdio.h>

29013 void printname(uid_t uid)
29014 {
29015     struct passwd *pwd;

29016     setpwent();
29017     while((pwd = getpwent()) != NULL) {
29018         if (pwd->pw_uid == uid) {
29019             printf("name=%s\n",pwd->pw_name);
29020             break;
29021         }
29022     }
29023     endpwent();
29024 }
```

29025 **APPLICATION USAGE**

29026 These functions are provided due to their historical usage. Applications should avoid
29027 dependencies on fields in the password database, whether the database is a single file, or where
29028 in the file system name space the database resides. Applications should use *getpwuid*( )
29029 whenever possible because it avoids these dependencies.

29030 **RATIONALE**

29031 None.

29032 **FUTURE DIRECTIONS**

29033 None.

29034 **SEE ALSO**

29035 *endgrent*( ), *getlogin*( ), *getpwnam*( ), *getpwuid*( )

29036 XBD **<pwd.h>**

29037 **CHANGE HISTORY**

29038 First released in Issue 4, Version 2.

29039 **Issue 5**

29040 Moved from X/OPEN UNIX extension to BASE.

29041 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
29042 VALUE section.

29043 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

Issue 6

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

The EXAMPLES section is revised.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0087 [75] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0092 [493], XSH/TC2-2008/0093 [656], and XSH/TC2-2008/0094 [493] are applied.

NAME

endservent, getservbyname, getservbyport, getservent, setservent — network services database functions

SYNOPSIS

```
#include <netdb.h>

void endservent(void);
struct servent *getservbyname(const char *name, const char *proto);
struct servent *getservbyport(int port, const char *proto);
struct servent *getservent(void);
void setservent(int stayopen);
```

DESCRIPTION

These functions shall retrieve information about network services. This information is considered to be stored in a database that can be accessed sequentially or randomly. The implementation of this database is unspecified.

The *setservent*( ) function shall open a connection to the database, and set the next entry to the first entry. If the *stayopen* argument is non-zero, the *net* database shall not be closed after each call to the *getservent*( ) function (either directly, or indirectly through one of the other *getserv**( ) functions), and the implementation may maintain an open file descriptor for the database. If a file descriptor is used, the FD_CLOEXEC flag shall be set; see **<fcntl.h>**.

The *getservent*( ) function shall read the next entry of the database, opening and closing a connection to the database as necessary.

The *getservbyname*( ) function shall search the database from the beginning and find the first entry for which the service name specified by *name* matches the *s_name* member and the protocol name specified by *proto* matches the *s_proto* member, opening and closing a connection to the database as necessary. If *proto* is a null pointer, any value of the *s_proto* member shall be matched.

The *getservbyport*( ) function shall search the database from the beginning and find the first entry for which the port specified by *port* matches the *s_port* member and the protocol name specified by *proto* matches the *s_proto* member, opening and closing a connection to the database as necessary. If *proto* is a null pointer, any value of the *s_proto* member shall be matched. The *port* argument shall be a value obtained by converting a **uint16_t** in network byte order to **int**.

The *getservbyname*( ), *getservbyport*( ), and *getservent*( ) functions shall each return a pointer to a **servent** structure, the members of which shall contain the fields of an entry in the network services database.

The *endservent*( ) function shall close the database, releasing any open file descriptor.

These functions need not be thread-safe.

RETURN VALUE

Upon successful completion, *getservbyname*( ), *getservbyport*( ), and *getservent*( ) return a pointer to a **servent** structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer is returned.

The application shall not modify the structure to which the return value points, nor any storage areas pointed to by pointers within the structure. The returned pointer, and pointers within the structure, might be invalidated or the structure or the storage areas might be overwritten by a subsequent call to *getservbyname*( ), *getservbyport*( ), or *getservent*( ). The returned pointer, and pointers within the structure, might also be invalidated if the calling thread is terminated.

ERRORS

The *getservbyname*( ), *getservbyport*( ), *getservent*( ), and *setservent*( ) functions may fail if:

[EMFILE] All file descriptors available to the process are currently open.

[ENFILE] The maximum allowable number of files is currently open in the system.

EXAMPLES

None.

APPLICATION USAGE

The *port* argument of *getservbyport*( ) need not be compatible with the port values of all address families.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endhostent( ), *endprotoent*( ), *htonl*( ), *inet_addr*( )

XBD **<fcntl.h>**, **<netdb.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XBD-ERN-14 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0088 [75] and XSH/TC1-2008/0089 [75] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0095 [656] is applied.

Issue 8

Austin Group Defect 368 is applied, adding a requirement for FD_CLOEXEC to be set if a file descriptor is used, and adding the [EMFILE] and [ENFILE] errors.

NAME

endutxent, getutxent, getutxid, getutxline, pututxline, setutxent — user accounting database functions

SYNOPSIS

XSI

```
#include <utmpx.h>

void endutxent(void);
struct utmpx *getutxent(void);
struct utmpx *getutxid(const struct utmpx *id);
struct utmpx *getutxline(const struct utmpx *line);
struct utmpx *pututxline(const struct utmpx *utmpx);
void setutxent(void);
```

DESCRIPTION

These functions shall provide access to the user accounting database.

The *getutxent*( ) function shall read the next entry from the user accounting database. If the database is not already open, it shall open it. If it reaches the end of the database, it shall fail.

The *getutxid*( ) function shall search forward from the current point in the database. If the *ut_type* value of the **utmpx** structure pointed to by *id* is BOOT_TIME, OLD_TIME, or NEW_TIME, then it shall stop when it finds an entry with a matching *ut_type* value. If the *ut_type* value is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then it shall stop when it finds an entry whose type is one of these four and whose *ut_id* member matches the *ut_id* member of the **utmpx** structure pointed to by *id*. If the end of the database is reached without a match, *getutxid*( ) shall fail.

The *getutxline*( ) function shall search forward from the current point in the database until it finds an entry of the type LOGIN_PROCESS or USER_PROCESS which also has a *ut_line* value matching that in the **utmpx** structure pointed to by *line*. If the end of the database is reached without a match, *getutxline*( ) shall fail.

The *getutxid*( ) or *getutxline*( ) function may cache data. For this reason, to use *getutxline*( ) to search for multiple occurrences, the application shall zero out the static data after each success, or *getutxline*( ) may return a pointer to the same **utmpx** structure.

There is one exception to the rule about clearing the structure before further reads are done. The implicit read done by *pututxline*( ) (if it finds that it is not already at the correct place in the user accounting database) shall not modify the static structure returned by *getutxent*( ), *getutxid*( ), or *getutxline*( ), if the application has modified this structure and passed the pointer back to *pututxline*( ).

For all entries that match a request, the *ut_type* member indicates the type of the entry. Other members of the entry shall contain meaningful data based on the value of the *ut_type* member as follows:

ut_type Member	Other Members with Meaningful Data
EMPTY	No others
BOOT_TIME	*ut_tv*
OLD_TIME	*ut_tv*
NEW_TIME	*ut_tv*
USER_PROCESS	*ut_id*, *ut_user* (login name of the user), *ut_line*, *ut_pid*, *ut_tv*
INIT_PROCESS	*ut_id*, *ut_pid*, *ut_tv*
LOGIN_PROCESS	*ut_id*, *ut_user* (implementation-defined name of the login process), *ut_line*, *ut_pid*, *ut_tv*
DEAD_PROCESS	*ut_id*, *ut_pid*, *ut_tv*

An implementation that provides extended security controls may impose implementation-defined restrictions on accessing the user accounting database. In particular, the system may deny the existence of some or all of the user accounting database entries associated with users other than the caller.

If the process has appropriate privileges, the *pututxline*( ) function shall write out the structure into the user accounting database. It shall search for a record as if by *getutxid*( ) that satisfies the request. If this search succeeds, then the entry shall be replaced. Otherwise, a new entry shall be made at the end of the user accounting database.

The *endutxent*( ) function shall close the user accounting database.

The *setutxent*( ) function shall reset the input to the beginning of the database. This should be done before each search for a new entry if it is desired that the entire database be examined.

These functions need not be thread-safe.

RETURN VALUE

Upon successful completion, *getutxent*( ), *getutxid*( ), and *getutxline*( ) shall return a pointer to a **utmpx** structure containing a copy of the requested entry in the user accounting database. Otherwise, a null pointer shall be returned.

The return value may point to a static area which is overwritten by a subsequent call to *getutxid*( ) or *getutxline*( ).

Upon successful completion, *pututxline*( ) shall return a pointer to a **utmpx** structure containing a copy of the entry added to the user accounting database. Otherwise, a null pointer shall be returned.

The *endutxent*( ) and *setutxent*( ) functions shall not return a value.

ERRORS

No errors are defined for the *endutxent*( ), *getutxent*( ), *getutxid*( ), *getutxline*( ), and *setutxent*( ) functions.

The *pututxline*( ) function may fail if:

[EPERM] The process does not have appropriate privileges.

29199 **EXAMPLES**

29200 None.

29201 **APPLICATION USAGE**

29202 The sizes of the arrays in the structure can be found using the *sizeof* operator.

29203 **RATIONALE**

29204 None.

29205 **FUTURE DIRECTIONS**

29206 None.

29207 **SEE ALSO**

29208 XBD **<utmpx.h>**

29209 **CHANGE HISTORY**

29210 First released in Issue 4, Version 2.

29211 **Issue 5**

29212 Moved from X/OPEN UNIX extension to BASE.

29213 Normative text previously in the APPLICATION USAGE section is moved to the
29214 DESCRIPTION.

29215 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

29216 **Issue 6**

29217 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

29218 **Issue 7**

29219 Austin Group Interpretation 1003.1-2001 #156 is applied.

29220 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0090 [213,428] and
29221 XSH/TC1-2008/0091 [213] are applied.

29222 **NAME**

29223 environ — array of character pointers to the environment strings

29224 **SYNOPSIS**

29225 `extern char **environ;`

29226 **DESCRIPTION**

29227 Refer to *exec* and XBD Chapter 8 (on page 167).

NAME

erand48 — generate uniformly distributed pseudo-random numbers

SYNOPSIS

XSI

```
#include <stdlib.h>

double erand48(unsigned short xsubi[3]);
```

DESCRIPTION

Refer to *drand48( )*.

NAME

erf, erff, erfl — error functions

SYNOPSIS

```
#include <math.h>

double erf(double x);
float erff(float x);
long double erfl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the error function of their argument x, defined as:

$$\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the value of the error function.

MX If x is NaN, a NaN shall be returned.

If x is ±0, ±0 shall be returned.

If x is ±Inf, ±1 shall be returned.

If the correct value would cause underflow, a range error may occur, and *erf*( ), *erff*( ), and *erfl*( ) shall return an implementation-defined value no greater in magnitude than DBL_MIN, MXX FLT_MIN, and LDBL_MIN, respectively. If the IEC 60559 Floating-Point option is supported, $2 * x / sqrt(\pi)$ should be returned.

ERRORS

These functions may fail if:

MX

Range Error	The result underflows. If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

Computing the Probability for a Normal Variate

This example shows how to use *erf*( ) to compute the probability that a normal variate assumes a value in the range [*x*1,*x*2] with *x*1≤*x*2.

This example uses the constant M_SQRT1_2 which is part of the XSI option.

```
#include <math.h>

double
Phi(const double x1, const double x2)
{
    return ( erf(x2*M_SQRT1_2) - erf(x1*M_SQRT1_2) ) / 2;
}
```

APPLICATION USAGE

Underflow occurs when $|x| <$ DBL_MIN * ($sqrt(\pi)/2$).

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

erfc( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *erf*( ) function is no longer marked as an extension.

The *erfc*( ) function is split out onto its own reference page.

The *erff*( ) and *erfl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/22 is applied, adding the example to the EXAMPLES section.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0092 [68] is applied.

29307 **Issue 8**

29308 Austin Group Defect 1178 is applied, joining two paragraphs in the RETURN VALUE section.

NAME

erfc, erfcf, erfcl — complementary error functions

SYNOPSIS

```
#include <math.h>

double erfc(double x);
float erfcf(float x);
long double erfcl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the complementary error function 1.0 – *erf* (*x*).

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the value of the complementary error function.

MXX If the correct value would cause underflow, and is not representable, a range error may occur, and *erfc*( ), *erfcf*( ), and *erfcl*( ) shall return 0.0, or (if the IEC 60559 Floating-Point option is not supported) an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, +1 shall be returned.

If *x* is –Inf, +2 shall be returned.

If *x* is +Inf, +0 shall be returned.

MXX If the correct value would cause underflow and is representable, a range error may occur and the correct value shall be returned.

ERRORS

These functions may fail if:

Range Error — The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

The *erfc*( ) function is provided because of the extreme loss of relative accuracy if *erf*(*x*) is called for large *x* and the result subtracted from 1.0.

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

erf( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *erfc*( ) function is no longer marked as an extension.

These functions are split out from the *erf*( ) reference page.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0093 [68] and XSH/TC1-2008/0094 [68] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0096 [630] is applied.

29375 **NAME**

29376 erff, erfl — error functions

29377 **SYNOPSIS**

29378 `#include <math.h>`

29379 `float erff(float x);`

29380 `long double erfl(long double x);`

29381 **DESCRIPTION**

29382 Refer to *erf( )*.

29383 **NAME**

29384 errno — error return value

29385 **SYNOPSIS**

29386 `#include <errno.h>`

29387 **DESCRIPTION**

29388 The lvalue to which the macro *errno* expands is used by many functions to return error values.

29389 Many functions provide an error number in *errno*, which has type **int** and is defined in
29390 **<errno.h>**. The value of *errno* in the initial thread shall be zero at program startup (the initial
29391 value of *errno* in other threads is an indeterminate value) and shall otherwise be defined only
29392 after a call to a function for which it is explicitly stated to be set and until it is changed by the
29393 next function call or if the application assigns it a value. The value of *errno* should only be
29394 examined when it is indicated to be valid by a function's return value. Applications shall obtain
29395 the definition of *errno* by the inclusion of **<errno.h>**. No function in this volume of
29396 POSIX.1-2024 shall set *errno* to 0. The setting of *errno* after a successful call to a function is
29397 unspecified unless the description of that function specifies that *errno* shall not be modified.

29398 If the macro definition is suppressed in order to access an actual object, or a program defines an
29399 identifier with the name *errno*, the behavior is undefined.

29400 The symbolic values stored in *errno* are documented in the ERRORS sections on all relevant
29401 pages.

29402 **RETURN VALUE**

29403 None.

29404 **ERRORS**

29405 None.

29406 **EXAMPLES**

29407 None.

29408 **APPLICATION USAGE**

29409 Previously both POSIX and X/Open documents were more restrictive than the ISO C standard
29410 in that they required *errno* to be defined as an external variable, whereas the ISO C standard
29411 required only that *errno* be defined as a modifiable lvalue with type **int**.

29412 An application that needs to examine the value of *errno* to determine the error should set it to 0
29413 before a function call, then inspect it before a subsequent function call.

29414 **RATIONALE**

29415 None.

29416 **FUTURE DIRECTIONS**

29417 None.

29418 **SEE ALSO**

29419 Section 2.3

29420 XBD **<errno.h>**

29421 **CHANGE HISTORY**

29422 First released in Issue 1. Derived from Issue 1 of the SVID.

29423 **Issue 5**

29424 The following sentence is deleted from the DESCRIPTION: ``The value of *errno* is 0 at program
29425 start-up, but is never set to 0 by any XSI function''.

The DESCRIPTION also no longer states that conforming implementations may support the declaration:

```
extern int errno;
```

Issue 6

Obsolescent text regarding defining *errno* as:

```
extern int errno
```

is removed.

Text regarding no function setting *errno* to zero to indicate an error is changed to no function shall set *errno* to zero. This is for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/23 is applied, adding text to the DESCRIPTION stating that the setting of *errno* after a successful call to a function is unspecified unless the description of the function requires that it will not be modified.

Issue 8

Austin Group Defect 1302 is applied, aligning this macro with the ISO/IEC 9899: 2018 standard.

NAME

environ, execl, execle, execlp, execv, execve, execvp, fexecve — execute a file

SYNOPSIS

```
#include <unistd.h>

extern char **environ;
int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
int execle(const char *path, const char *arg0, ... /*,
    (char *)0, char *const envp[]*/);
int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
int execv(const char *path, char *const argv[]);
int execve(const char *path, char *const argv[], char *const envp[]);
int execvp(const char *file, char *const argv[]);
int fexecve(int fd, char *const argv[], char *const envp[]);
```

DESCRIPTION

The *exec* family of functions shall replace the current process image with a new process image. The new image shall be constructed from a regular, executable file called the *new process image file*. There shall be no return from a successful *exec*, because the calling process image is overlaid by the new process image.

The *fexecve*() function shall be equivalent to the *execve*() function except that the file to be executed is determined by the file descriptor *fd* instead of a pathname. The file offset of *fd* is ignored.

When a C-language program is executed as a result of a call to one of the *exec* family of functions, it shall be entered as a C-language function call as follows:

```
int main (int argc, char *argv[]);
```

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. In addition, the following variable, which needs to be declared by the user if it is to be used directly:

```
extern char **environ;
```

is initialized as a pointer to an array of character pointers to the environment strings. The *argv* and *environ* arrays are each terminated by a null pointer. The null pointer terminating the *argv* array is not counted in *argc*.

Applications can change the entire environment in a single operation by assigning the *environ* variable to point to an array of character pointers to the new environment strings. After assigning a new value to *environ*, applications should not rely on the new environment strings

XSI remaining part of the environment, as a call to *getenv*(), *secure_getenv*(), *putenv*(), *setenv*(), *unsetenv*(), or any function that is dependent on an environment variable may, on noticing that *environ* has changed, copy the environment strings to a new array and assign *environ* to point to it.

Any application that directly modifies the pointers to which the *environ* variable points has undefined behavior.

Conforming multi-threaded applications shall not use the *environ* variable to access or modify any environment variable while any other thread is concurrently modifying any environment variable. A call to any function dependent on any environment variable shall be considered a use of the *environ* variable to access that environment variable.

The arguments specified by a program with one of the *exec* functions shall be passed on to the new process image in the corresponding *main*() arguments.

For the *execl*( ), *execle*( ), *execv*( ), and *execve*( ) functions, the argument *path* points to a pathname that identifies the new process image file.

For the *execlp*( ) and *execvp*( ) functions, the argument *file* is used to construct a pathname that identifies the new process image file. If the *file* argument contains a <slash> character, the *file* argument shall be used as the pathname for this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed as the environment variable *PATH* (see XBD Chapter 8, on page 167). If this environment variable is not present, the results of the search are implementation-defined.

There are two distinct ways in which the contents of the process image file may cause the execution to fail, distinguished by the setting of *errno* to either [ENOEXEC] or [EINVAL] (see the ERRORS section). In the cases where the other members of the *exec* family of functions would fail and set *errno* to [ENOEXEC], the *execlp*( ) and *execvp*( ) functions shall execute a command interpreter and the environment of the executed command shall be as if the process invoked the *sh* utility using *execl*( ) as follows:

```
execl(<shell path>, <name>, file, <args>, (char *)0);
```

where <*shell path*> is an unspecified pathname for the *sh* utility, <*name*> is an unspecified string, *file* is the process image file, and where <*args*> is zero or more parameters corresponding to any initial non-null arguments passed after *arg0* for *execlp*( ) or to any initial non-null members of *argv* starting at *argv*[1] for *execvp*( ).

The arguments represented by *arg0*,... are pointers to null-terminated character strings. These strings shall constitute the argument list available to the new process image. The list is terminated by a null pointer. The argument *arg0* should point to a filename string that is associated with the process being started by one of the *exec* functions.

The argument *argv* is an array of character pointers to null-terminated strings. The application shall ensure that the last member of this array is a null pointer. These strings shall constitute the argument list available to the new process image. The value in *argv*[0] should point to a filename string that is associated with the process being started by one of the *exec* functions.

The argument *envp* is an array of character pointers to null-terminated strings. These strings shall constitute the environment for the new process image. The *envp* array is terminated by a null pointer.

For those forms not containing an *envp* pointer (*execl*( ), *execv*( ), *execlp*( ), and *execvp*( )), the environment for the new process image shall be taken from the external variable *environ* in the calling process.

The number of bytes available for the new process' combined argument and environment lists is {ARG_MAX}. It is implementation-defined whether null terminators, pointers, and/or any alignment bytes are included in this total.

File descriptors open in the calling process image shall remain open in the new process image, except for those whose close-on-*exec* flag FD_CLOEXEC is set. For those file descriptors that remain open, all attributes of the open file description shall remain unchanged and the FD_CLOFORK file descriptor flag, if set, shall remain set. For any file descriptor that is closed for this reason, process-owned file locks that the calling process owns on the file associated with the file descriptor shall be unlocked as a result of the close, as described in *close*( ). File locks that are not unlocked by closing of file descriptors remain unchanged.

If file descriptor 0, 1, or 2 would otherwise be closed after a successful call to one of the *exec* family of functions, implementations may open an unspecified file for the file descriptor in the new process image. If a standard utility or a conforming application is executed with file

29532 descriptor 0 not open for reading or with file descriptor 1 or 2 not open for writing, the
29533 environment in which the utility or application is executed shall be deemed non-conforming,
29534 and consequently the utility or application might not behave as described in this standard.

29535 Directory streams open in the calling process image shall be closed in the new process image.

29536 The state of the floating-point environment in the initial thread of the new process image shall
29537 be set to the default.

29538 The state of conversion descriptors and message catalog descriptors in the new process image is
29539 undefined.

29540 For the new process image, the equivalent of:

29541 `setlocale(LC_ALL, "C")`

29542 shall be executed at start-up.

29543 Signals set to the default action (SIG_DFL) in the calling process image shall be set to the default
29544 action in the new process image. Except for SIGCHLD, signals set to be ignored (SIG_IGN) by
29545 the calling process image shall be set to be ignored by the new process image. Signals set to be
29546 caught by the calling process image shall be set to the default action in the new process image
29547 (see **<signal.h>**).

29548 If the SIGCHLD signal is set to be ignored by the calling process image, it is unspecified whether
29549 the SIGCHLD signal is set to be ignored or to the default action in the new process image.

29550 XSI After a successful call to any of the *exec* functions, alternate signal stacks are not preserved and
29551 the SA_ONSTACK flag shall be cleared for all signals.

29552 After a successful call to any of the *exec* functions, any functions previously registered by the
29553 *atexit*( ), *at_quick_exit*( ), or *pthread_atfork*( ) functions are no longer registered.

29554 XSI If the ST_NOSUID bit is set for the file system containing the new process image file, then the
29555 effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged
29556 in the new process image. Otherwise, if the set-user-ID mode bit of the new process image file is
29557 set, the effective user ID of the new process image shall be set to the user ID of the new process
29558 image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the
29559 effective group ID of the new process image shall be set to the group ID of the new process
29560 image file. The real user ID, real group ID, and supplementary group IDs of the new process
29561 image shall remain the same as those of the calling process image. The effective user ID and
29562 effective group ID of the new process image shall be saved (as the saved set-user-ID and the
29563 saved set-group-ID) for use by *setuid*( ).

29564 XSI Any shared memory segments attached to the calling process image shall not be attached to the
29565 new process image.

29566 Any named semaphores open in the calling process shall be closed as if by appropriate calls to
29567 *sem_close*( ). Any unnamed semaphores open in the calling process shall be destroyed as if by
29568 calls to *sem_destroy*( ).

29569 TYM Any blocks of typed memory that were mapped in the calling process are unmapped, as if
29570 *munmap*( ) was implicitly called to unmap them.

29571 ML Memory locks established by the calling process via calls to *mlockall*( ) or *mlock*( ) shall be
29572 removed. If locked pages in the address space of the calling process are also mapped into the
29573 address spaces of other processes and are locked by those processes, the locks established by the
29574 other processes shall be unaffected by the call by this process to the *exec* function. If the *exec*
29575 function fails, the effect on memory locks is unspecified.

Memory mappings created in the process are unmapped before the address space is rebuilt for the new process image.

SS When the calling process image does not use the SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC scheduling policies, the scheduling policy and parameters of the new process image and the initial thread in that new process image are implementation-defined.

PS When the calling process image uses the SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC scheduling policies, the process policy and scheduling parameter settings shall not be changed by a call to an *exec* function. TPS The initial thread in the new process image shall inherit the process scheduling policy and parameters. It shall have the default system contention scope, but shall inherit its allocation domain from the calling process image.

Per-process timers created by the calling process shall be deleted before replacing the current process image with the new process image.

MSG All open message queue descriptors in the calling process shall be closed, as described in *mq_close*( ).

Any outstanding asynchronous I/O operations may be canceled. Those asynchronous I/O operations that are not canceled shall complete as if the *exec* function had not yet occurred, but any associated signal notifications shall be suppressed. It is unspecified whether the *exec* function itself blocks awaiting such I/O completion. In no event, however, shall the new process image created by the *exec* function be affected by the presence of outstanding asynchronous I/O operations at the time the *exec* function is called. Whether any I/O is canceled, and which I/O may be canceled upon *exec*, is implementation-defined.

CPT The new process image shall inherit the CPU-time clock of the calling process image. This inheritance means that the process CPU-time clock of the process being *exec*-ed shall not be reinitialized or altered as a result of the *exec* function other than to reflect the time spent by the process executing the *exec* function itself.

TCT The initial value of the CPU-time clock of the initial thread of the new process image shall be set to zero.

The thread ID of the initial thread in the new process image is unspecified.

The size and location of the stack on which the initial thread in the new process image runs is unspecified.

The initial thread in the new process image shall have its cancellation type set to PTHREAD_CANCEL_DEFERRED and its cancellation state set to PTHREAD_CANCEL_ENABLED.

The initial thread in the new process image shall have all thread-specific data values set to NULL and all thread-specific data keys shall be removed by the call to *exec* without running destructors.

The initial thread in the new process image shall be joinable, as if created with the *detachstate* attribute set to PTHREAD_CREATE_JOINABLE.

The new process shall inherit at least the following attributes from the calling process image:

- XSI Nice value (see *nice*( ))
- XSI *semadj* values (see *semop*( ))
- Process ID

- Parent process ID
- Process group ID
- Session membership
- Real user ID
- Real group ID
- Supplementary group IDs
- Time left until an alarm clock signal (see *alarm*( ))
- Current working directory
- Root directory
- File mode creation mask (see *umask*( ))
- File size limit (see *getrlimit*( ) and *setrlimit*( ))
- Process signal mask (see *pthread_sigmask*( ))
- Pending signal (see *sigpending*( ))
- *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime* (see *times*( ))
- Resource limits
- Controlling terminal

The initial thread of the new process shall inherit at least the following attributes from the calling thread:

- Signal mask (see *sigprocmask*( ) and *pthread_sigmask*( ))
- Pending signals (see *sigpending*( ))

All other process attributes defined in this volume of POSIX.1-2024 shall be inherited in the new process image from the old process image. All other thread attributes defined in this volume of POSIX.1-2024 shall be inherited in the initial thread in the new process image from the calling thread in the old process image. The inheritance of process or thread attributes not defined by this volume of POSIX.1-2024 is implementation-defined.

A call to any *exec* function from a process with more than one thread shall result in all threads being terminated and the new executable image being loaded and executed. No destructor functions or cleanup handlers shall be called.

Upon successful completion, the *exec* functions shall mark for update the last data access timestamp of the file. If an *exec* function failed but was able to locate the process image file, whether the last data access timestamp is marked for update is unspecified. Should the *exec* function succeed, the process image file shall be considered to have been opened with *open*( ). The corresponding *close*( ) shall be considered to occur at a time after this open, but before process termination or successful completion of a subsequent call to one of the *exec* functions, *posix_spawn*( ), or *posix_spawnp*( ). The *argv*[ ] and *envp*[ ] arrays of pointers and the strings to which those arrays point shall not be modified by a call to one of the *exec* functions, except as a consequence of replacing the process image.

The saved resource limits in the new process image are set to be a copy of the process' corresponding hard and soft limits.

29657 **RETURN VALUE**

29658 If one of the *exec* functions returns to the calling process image, an error has occurred; the return
29659 value shall be –1, and *errno* shall be set to indicate the error.

29660 **ERRORS**

29661 The *exec* functions shall fail if:

29662 [E2BIG] The number of bytes used by the new process image's argument list and
29663 environment list is greater than the system-imposed limit of {ARG_MAX}
29664 bytes.

29665 [EACCES] The new process image file is not a regular file and the implementation does
29666 not support execution of files of its type.

29667 [EINVAL] The new process image file has appropriate privileges and has a recognized
29668 executable binary format, but the system does not support execution of a file
29669 with this format.

29670 The *exec* functions, except for *fexecve*( ), shall fail if:

29671 [EACCES] Search permission is denied for a directory listed in the new process image
29672 file's path prefix, or the new process image file denies execution permission.

29673 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* or *file*
29674 argument.

29675 [ENAMETOOLONG]
29676 The length of a component of a pathname is longer than {NAME_MAX}.

29677 [ENOENT] A component of *path* or *file* does not name an existing file or *path* or *file* is an
29678 empty string.

29679 [ENOTDIR] A component of the new process image file's path prefix names an existing file
29680 that is neither a directory nor a symbolic link to a directory, or the new process
29681 image file's pathname contains at least one non-<slash> character and ends
29682 with one or more trailing <slash> characters and the last pathname
29683 component names an existing file that is neither a directory nor a symbolic
29684 link to a directory.

29685 The *exec* functions, except for *execlp*( ) and *execvp*( ), shall fail if:

29686 [ENOEXEC] The new process image file has the appropriate access permission but has an
29687 unrecognized format.

29688 The *fexecve*( ) function shall fail if:

29689 [EBADF] The *fd* argument is not a valid file descriptor open for executing.

29690 The *exec* functions may fail if:

29691 [ENOMEM] The new process image requires more memory than is allowed by the
29692 hardware or system-imposed memory management constraints.

29693 The *exec* functions, except for *fexecve*( ), may fail if:

29694 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
29695 resolution of the *path* or *file* argument.

29696 [ENAMETOOLONG]
29697 The length of the *path* argument or the length of the pathname constructed
29698 from the *file* argument exceeds {PATH_MAX}, or pathname resolution of a

symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

[ETXTBSY] The new process image file is a pure procedure (shared text) file that is currently open for writing by some process.

EXAMPLES

Using execl()

The following example executes the *ls* command, specifying the pathname of the executable (**/bin/ls**) and using arguments supplied directly to the command to produce single-column output.

```
#include <unistd.h>

int ret;
...
ret = execl ("/bin/ls", "ls", "-1", (char *)0);
```

Using execle()

The following example is similar to Using execl(). In addition, it specifies the environment for the new process image using the *env* argument.

```
#include <unistd.h>

int ret;
char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
...
ret = execle ("/bin/ls", "ls", "-l", (char *)0, env);
```

Using execlp()

The following example searches for the location of the *ls* command among the directories specified by the *PATH* environment variable.

```
#include <unistd.h>

int ret;
...
ret = execlp ("ls", "ls", "-l", (char *)0);
```

Using execv()

The following example passes arguments to the *ls* command in the *cmd* array.

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
...
ret = execv ("/bin/ls", cmd);
```

29734 **Using execve()**

29735 The following example passes arguments to the *ls* command in the *cmd* array, and specifies the
29736 environment for the new process image using the *env* argument.

```
29737 #include <unistd.h>

29738 int ret;
29739 char *cmd[] = { "ls", "-l", (char *)0 };
29740 char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
29741 ...
29742 ret = execve ("/bin/ls", cmd, env);
```

29743 **Using execvp()**

29744 The following example searches for the location of the *ls* command among the directories
29745 specified by the *PATH* environment variable, and passes arguments to the *ls* command in the
29746 *cmd* array.

```
29747 #include <unistd.h>

29748 int ret;
29749 char *cmd[] = { "ls", "-l", (char *)0 };
29750 ...
29751 ret = execvp ("ls", cmd);
```

29752 **APPLICATION USAGE**

29753 As the state of conversion descriptors and message catalog descriptors in the new process image
29754 is undefined, conforming applications should not rely on their use and should close them prior
29755 to calling one of the *exec* functions.

29756 Applications that require other than the default POSIX locale as the global locale in the new
29757 process image should call *setlocale*() with the appropriate parameters.

29758 When assigning a new value to the *environ* variable, applications should ensure that the
29759 environment to which it will point contains at least the following:

29760 1. Any implementation-defined variables required by the implementation to provide a
29761 conforming environment. See the _CS_V8_ENV entry in **<unistd.h>** and *confstr*() for
29762 details.

29763 2. A value for *PATH* which finds conforming versions of all standard utilities before any
29764 other versions.

29765 The same constraint applies to the *envp* array passed to *execle*() or *execve*(), in order to ensure
29766 that the new process image is invoked in a conforming environment.

29767 Applications should not execute programs with file descriptor 0 not open for reading or with file
29768 descriptor 1 or 2 not open for writing, as this might cause the executed program to misbehave.
29769 In order not to pass on these file descriptors to an executed program, applications should not
29770 just close them but should reopen them on, for example, **/dev/null**. Some implementations may
29771 reopen them automatically, but applications should not rely on this being done.

29772 If an application wants to perform an integrity test of the file being executed before executing it,
29773 the file will need to be opened with read permission to perform the integrity test.

29774 Since execute permission is checked by *fexecve*(), the file description *fd* need not have been
29775 opened with the O_EXEC flag. However, if the file to be executed denies read and write
29776 permission for the process preparing to do the *exec*, the only way to provide the *fd* to *fexecve*()

will be to use the O_EXEC flag when opening *fd*. In this case, the application will not be able to perform an integrity test since it will not be able to read the contents of the file.

Note that when a file descriptor is opened with O_RDONLY, O_RDWR, or O_WRONLY mode, the file descriptor can be used to read, read and write, or write the file, respectively, even if the mode of the file changes after the file was opened. Using the O_EXEC open mode is different; *fexecve*() will ignore the mode that was used when the file descriptor was opened and the *exec* will fail if the mode of the file associated with *fd* does not grant execute permission to the calling process at the time *fexecve*() is called.

RATIONALE

Early proposals required that the value of *argc* passed to *main*() be ``one or greater''. This was driven by the same requirement in drafts of the ISO C standard. In fact, historical implementations have passed a value of zero when no arguments are supplied to the caller of the *exec* functions. This requirement was removed from the ISO C standard and subsequently removed from this volume of POSIX.1-2024 as well. The wording, in particular the use of the word *should*, requires a Strictly Conforming POSIX Application to pass at least one argument to the *exec* function, thus guaranteeing that *argc* be one or greater when invoked by such an application. In fact, this is good practice, since many existing applications reference *argv*[0] without first checking the value of *argc*.

The requirement on a Strictly Conforming POSIX Application also states that the value passed as the first argument be a filename string associated with the process being started. Although some existing applications pass a pathname rather than a filename string in some circumstances, a filename string is more generally useful, since the common usage of *argv*[0] is in printing diagnostics. In some cases the filename passed is not the actual filename of the file; for example, many implementations of the *login* utility use a convention of prefixing a <hyphen-minus> (`'-'`) to the actual filename, which indicates to the command interpreter being invoked that it is a ``login shell''.

Also, note that the *test* and *[* utilities require specific strings for the *argv*[0] argument to have deterministic behavior across all implementations.

Historically, there have been two ways that implementations can *exec* shell scripts.

One common historical implementation is that the *execl*(), *execv*(), *execle*(), and *execve*() functions return an [ENOEXEC] error for any file not recognizable as executable, including a shell script. When the *execlp*() and *execvp*() functions encounter such a file, they assume the file to be a shell script and invoke a known command interpreter to interpret such files. This is now required by POSIX.1-2024. These implementations of *execvp*() and *execlp*() only give the [ENOEXEC] error in the rare case of a problem with the command interpreter's executable file. Because of these implementations, the [ENOEXEC] error is not mentioned for *execlp*() or *execvp*(), although implementations can still give it.

Another way that some historical implementations handle shell scripts is by recognizing the first two bytes of the file as the character string `"#!"` and using the remainder of the first line of the file as the name of the command interpreter to execute.

One potential source of confusion noted by the standard developers is over how the contents of a process image file affect the behavior of the *exec* family of functions. The following is a description of the actions taken:

1. If the process image file is a valid executable (in a format that is executable and valid and having appropriate privileges) for this system, then the system executes the file.

2. If the process image file has appropriate privileges and is in a format that is executable but not valid for this system (such as a recognized binary for another architecture), then this is an error and *errno* is set to [EINVAL] (see later RATIONALE on [EINVAL]).
3. If the process image file has appropriate privileges but is not otherwise recognized:
    a. If this is a call to *execlp*() or *execvp*(), then they invoke a command interpreter assuming that the process image file is a shell script.
    b. If this is not a call to *execlp*() or *execvp*(), then an error occurs and *errno* is set to [ENOEXEC].

Applications that do not require to access their arguments may use the form:

```
main(void)
```

as specified in the ISO C standard. However, the implementation will always provide the two arguments *argc* and *argv*, even if they are not used.

Some implementations provide a third argument to *main*() called *envp*. This is defined as a pointer to the environment. The ISO C standard specifies invoking *main*() with two arguments, so implementations are required to support applications written this way. Since this volume of POSIX.1-2024 defines the global variable *environ*, which is also provided by historical implementations and can be used anywhere that *envp* could be used, there is no functional need for the *envp* argument. Applications should use the *getenv*() function rather than accessing the environment directly via either *envp* or *environ*. Implementations are required to support the two-argument calling sequence, but this does not prohibit an implementation from supporting *envp* as an optional third argument.

This volume of POSIX.1-2024 specifies that signals set to SIG_IGN remain set to SIG_IGN, and that the new process image inherits the signal mask of the thread that called *exec* in the old process image. This is consistent with historical implementations, and it permits some useful functionality, such as the *nohup* command. However, it should be noted that many existing applications wrongly assume that they start with certain signals set to the default action and/or unblocked. In particular, applications written with a simpler signal model that does not include blocking of signals, such as the one in the ISO C standard, may not behave properly if invoked with some signals blocked. Therefore, it is best not to block or ignore signals across *exec*s without explicit reason to do so, and especially not to block signals across *exec*s of arbitrary (not closely cooperating) programs.

The *exec* functions always save the value of the effective user ID and effective group ID of the process at the completion of the *exec*, whether or not the set-user-ID or the set-group-ID bit of the process image file is set.

The statement about *argv*[ ] and *envp*[ ] being constants is included to make explicit to future writers of language bindings that these objects are completely constant. Due to a limitation of the ISO C standard, it is not possible to state that idea in standard C. Specifying two levels of *const–qualification* for the *argv*[ ] and *envp*[ ] parameters for the *exec* functions may seem to be the natural choice, given that these functions do not modify either the array of pointers or the characters to which the function points, but this would disallow existing correct code. Instead, only the array of pointers is noted as constant. The table of assignment compatibility for *dst=src* derived from the ISO C standard summarizes the compatibility:

dst:	**char *[ ]**	**const char *[ ]**	**char *const[ ]**	**const char *const[ ]**
src:				
char *[ ]	VALID	—	VALID	—
const char *[ ]	—	VALID	—	VALID
char * const [ ]	—	—	VALID	—
const char *const[ ]	—	—	—	VALID

Since all existing code has a source type matching the first row, the column that gives the most valid combinations is the third column. The only other possibility is the fourth column, but using it would require a cast on the *argv* or *envp* arguments. It is unfortunate that the fourth column cannot be used, because the declaration a non-expert would naturally use would be that in the second row.

The ISO C standard and this volume of POSIX.1-2024 do not conflict on the use of *environ*, but some historical implementations of *environ* may cause a conflict. As long as *environ* is treated in the same way as an entry point (for example, *fork*()), it conforms to both standards. A library can contain *fork*(), but if there is a user-provided *fork*(), that *fork*() is given precedence and no problem ensues. The situation is similar for *environ*: the definition in this volume of POSIX.1-2024 is to be used if there is no user-provided *environ* to take precedence. At least three implementations are known to exist that solve this problem.

[E2BIG] The limit {ARG_MAX} applies not just to the size of the argument list, but to the sum of that and the size of the environment list.

[EFAULT] Some historical systems return [EFAULT] rather than [ENOEXEC] when the new process image file is corrupted. They are non-conforming.

[EINVAL] This error condition was added to POSIX.1-2024 to allow an implementation to detect executable files generated for different architectures, and indicate this situation to the application. Historical implementations of shells, *execvp*(), and *execlp*() that encounter an [ENOEXEC] error will execute a shell on the assumption that the file is a shell script. This will not produce the desired effect when the file is a valid executable for a different architecture. An implementation may now choose to avoid this problem by returning [EINVAL] when a valid executable for a different architecture is encountered. Some historical implementations return [EINVAL] to indicate that the *path* argument contains a character with the high order bit set. The standard developers chose to deviate from historical practice for the following reasons:

1. The new utilization of [EINVAL] will provide some measure of utility to the user community.
2. Historical use of [EINVAL] is not acceptable in an internationalized operating environment.

[ENAMETOOLONG] Since the file pathname may be constructed by taking elements in the *PATH* variable and putting them together with the filename, the [ENAMETOOLONG] error condition could also be reached this way.

[ETXTBSY] System V returns this error when the executable file is currently open for writing by some process. This volume of POSIX.1-2024 neither requires nor prohibits this behavior.

Other systems (such as System V) may return [EINTR] from *exec*. This is not addressed by this volume of POSIX.1-2024, but implementations may have a window between the call to *exec* and

the time that a signal could cause one of the *exec* calls to return with [EINTR].

An explicit statement regarding the floating-point environment (as defined in the **<fenv.h>** header) was added to make it clear that the floating-point environment is set to its default when a call to one of the *exec* functions succeeds. The requirements for inheritance or setting to the default for other process and thread start-up functions is covered by more generic statements in their descriptions and can be summarized as follows:

posix_spawn( )	Set to default.
fork( )	Inherit.
pthread_create( )	Inherit.

The purpose of the *fexecve*( ) function is to enable executing a file which has been verified to be the intended file. It is possible to actively check the file by reading from the file descriptor and be sure that the file is not exchanged for another between the reading and the execution. Alternatively, a function like *openat*( ) can be used to open a file which has been found by reading the content of a directory using *readdir*( ).

When *execlp*( ) or *execvp*( ) fall back to invoking *sh* because of an [ENOEXEC] condition, the standard leaves the process name (what becomes *argv*[0] in the resulting *sh* process) unspecified. Existing implementations vary on whether they pass a variation of `"sh"`, or preserve the original *arg0*. There are existing implementations of *sh* that behave differently depending on the contents of *argv*[0], such that blindly passing the original *arg0* on to the fallback execution can fail to invoke a compliant shell environment. Because of the requirements on how *sh* handles its command line arguments, the shell script will see `$0` containing the pathname of the script being executed, regardless of the value of *argv*[0].

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *atexit*( ), *chmod*( ), *close*( ), *confstr*( ), *exit*( ), *fcntl*( ), *fork*( ), *fstatvfs*( ), *getenv*( ), *getrlimit*( ), *mknod*( ), *mmap*( ), *nice*( ), *open*( ), *posix_spawn*( ), *pthread_atfork*( ), *pthread_sigmask*( ), *putenv*( ), *readdir*( ), *semop*( ), *setlocale*( ), *shmat*( ), *sigaction*( ), *sigaltstack*( ), *sigpending*( ), *system*( ), *times*( ), *umask*( )

XBD Chapter 8 (on page 167), **<unistd.h>**

XCU *test*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Large File Summit extensions are added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, behavior is defined for when the process image file is not a valid executable.
- In this version, _POSIX_SAVED_IDS is mandated, thus the effective user ID and effective group ID of the new process image shall be saved (as the saved set-user-ID and the saved set-group-ID) for use by the *setuid*( ) function.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [ETXTBSY] optional error condition is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [EINVAL] mandatory error condition is added.
- The [ELOOP] optional error condition is added.

The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for typed memory.

The normative text is updated to avoid use of the term ``must'' for application requirements.

The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

IEEE PASC Interpretation 1003.1 #132 is applied.

The DESCRIPTION is updated to make it explicit that the floating-point environment in the new process image is set to the default.

The DESCRIPTION and RATIONALE are updated to include clarifications of how the contents of a process image file affect the behavior of the *exec* functions.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/15 is applied, adding a new paragraph to the DESCRIPTION and text to the end of the APPLICATION USAGE section. This change addresses a security concern, where implementations may want to reopen file descriptors 0, 1, and 2 for programs with the set-user-id or set-group-id file mode bits calling the *exec* family of functions.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/24 is applied, applying changes to the DESCRIPTION, addressing which attributes are inherited by threads, and behavioral requirements for threads attributes.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/25 is applied, updating text in the RATIONALE from ``the process signal mask be unchanged across an *exec*'' to ``the new process image inherits the signal mask of the thread that called *exec* in the old process image''.

Issue 7

Austin Group Interpretation 1003.1-2001 #047 is applied, adding the description of _CS_V7_ENV to the APPLICATION USAGE.

Austin Group Interpretation 1003.1-2001 #143 is applied.

The *fexecve*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, Threads, and Timers options is moved to the Base.

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0095 [386], XSH/TC1-2008/0096 [167], XSH/TC1-2008/0097 [291], XSH/TC1-2008/0098 [173], XSH/TC1-2008/0099 [296], XSH/TC1-2008/00100 [324], XSH/TC1-2008/00101 [296], XSH/TC1-2008/00102 [302], XSH/TC1-2008/00103 [167], XSH/TC1-2008/00104 [173], and XSH/TC1-2008/00105 [291,429] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0097 [584], XSH/TC2-2008/0098 [898], and XSH/TC2-2008/0099 [734] are applied.

Issue 8

Austin Group Defects 51 and 1669 are applied, moving the *getrlimit*( ) and *setrlimit*( ) functions from the XSI option to the Base.

Austin Group Defect 368 is applied, adding a requirement for unnamed semaphores to be destroyed.

Austin Group Defect 768 is applied, adding OFD-owned file locks.

Austin Group Defect 922 is applied, adding the *secure_getenv*( ) function.

Austin Group Defect 1284 is applied, changing ``checksum test'' to ``integrity test'' in the APPLICATION USAGE section.

Austin Group Defect 1318 is applied, adding FD_CLOFORK.

Austin Group Defect 1330 is applied, removing obsolescent interfaces and changing ``_V7_'' to ``_V8_''.

Austin Group Defect 1435 is applied, adding function lists to the descriptions of the *path* and *file* arguments.

Austin Group Defect 1645 is applied, making it unspecified what string is passed to the shell in *argv*[0] when executed by *execlp*( ) or *execvp*( ).

Austin Group Defect 1646 is applied, adding *at_quick_exit*( ) to the list of registration functions whose registrations are not inherited across *exec*.

30016 **NAME**

30017 exit — terminate a process

30018 **SYNOPSIS**

30019 ```
#include <stdlib.h>
```

30020 ```
void exit(int status);
```

30021 **DESCRIPTION**

30022 CX The functionality described on this reference page is aligned with the ISO C standard. Any
30023 conflict between the requirements described here and the ISO C standard is unintentional. This
30024 volume of POSIX.1-2024 defers to the ISO C standard.

30025 The *exit*( ) function shall cause normal process termination to occur. No functions registered by
30026 the *at_quick_exit*( ) function shall be called. If a process calls the *exit*( ) function more than once,
30027 or calls the *quick_exit*( ) function in addition to the *exit*( ) function, the behavior is undefined.

30028 CX The value of *status* can be 0, EXIT_SUCCESS, EXIT_FAILURE, or any other value, though only
30029 the least significant 8 bits (that is, *status* & 0377) shall be available from *wait*( ) and *waitpid*( ); the
30030 full value shall be available from *waitid*( ) and in the **siginfo_t** passed to a signal handler for
30031 SIGCHLD.

30032 The *exit*( ) function shall first call all functions registered by *atexit*( ), in the reverse order of their
30033 registration, except that a function is called after any previously registered functions that had
30034 already been called at the time it was registered. Each function is called as many times as it was
30035 registered. If, during the call to any such function, a call to the *longjmp*( ) function is made that
30036 would terminate the call to the registered function, the behavior is undefined.

30037 If a function registered by a call to *atexit*( ) fails to return, the remaining registered functions shall
30038 not be called and the rest of the *exit*( ) processing shall not be completed.

30039 CX The *exit*( ) function shall then flush all open streams with unwritten buffered data. For each
30040 stream which is the active handle to its underlying file descriptor, and for which the file is not
30041 already at EOF and is capable of seeking, the file offset of the underlying open file description
30042 shall be set to the file position of the stream. For each open stream, the *exit*( ) function shall
30043 perform the equivalent of a *close*( ) on the file descriptor that is associated with the stream.
30044 CX Finally, the process shall be terminated with the same consequences as described in
30045 Consequences of Process Termination (on page 568).

30046 **RETURN VALUE**

30047 The *exit*( ) function does not return.

30048 **ERRORS**

30049 No errors are defined.

30050 **EXAMPLES**

30051 See APPLICATION USAGE.

30052 **APPLICATION USAGE**

30053 When a stream that has unwritten buffered data is flushed by *exit*( ) there is no way for the
30054 calling process to discover whether or not *exit*( ) successfully wrote the data to the underlying
30055 file descriptor. Therefore, it is strongly recommended that applications always ensure there is no
30056 unwritten buffered data in any stream when calling *exit*( ), or returning from the initial call to
30057 *main*( ), with a *status* value that indicates no errors occurred.

30058 For example, the following code demonstrates one way to ensure that *stdout* has already been
30059 successfully flushed before calling *exit*( ) with status 0. If the flush fails, the file descriptor
30060 underlying *stdout* is closed so that *exit*( ) will not try to repeat the failed write operation. If the
30061 flush succeeds, a final check with *ferror*( ) is performed to ensure that there were no write errors

during earlier flush operations (that were not handled at the time).

```
int status = 0;
if (fflush(stdout) != 0) {
    perror("appname: standard output");
    close(fileno(stdout));
    status = 1;
}
else if (ferror(stdout)) {
    fputs("appname: write error on standard output\n", stderr);
    status = 1;
}
exit(status);
```

See also *_Exit*( ).

RATIONALE

See *_Exit*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

_Exit( ), *at_quick_exit*( ), *atexit*( ), *exec*, *fflush*( ), *longjmp*( ), *quick_exit*( ), *tmpfile*( ), *wait*( ), *waitid*( )

XBD **<stdlib.h>**

CHANGE HISTORY

Issue 7

Austin Group Interpretation 1003.1-2001 #031 is applied, separating the *_Exit*( ) and *_exit*( ) functions from the *exit*( ) function.

Austin Group Interpretation 1003.1-2001 #085 is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0100 [594] is applied.

Issue 8

Austin Group Defect 610 is applied, clarifying the effects of *exit*( ) on open streams.

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018 standard.

Austin Group Defect 1490 is applied, changing the EXAMPLES and APPLICATION USAGE sections.

Austin Group Defect 1629 is applied, changing the APPLICATION USAGE section.

30095 **NAME**

30096 exp, expf, expl — exponential function

30097 **SYNOPSIS**

30098
```
#include <math.h>
```

30099
```
double exp(double x);
```
30100
```
float expf(float x);
```
30101
```
long double expl(long double x);
```

30102 **DESCRIPTION**

30103 CX The functionality described on this reference page is aligned with the ISO C standard. Any
30104 conflict between the requirements described here and the ISO C standard is unintentional. This
30105 volume of POSIX.1-2024 defers to the ISO C standard.

30106 These functions shall compute the base-*e* exponential of *x*.

30107 An application wishing to check for error situations should set *errno* to zero and call
30108 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
30109 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
30110 zero, an error has occurred.

30111 **RETURN VALUE**

30112 Upon successful completion, these functions shall return the exponential value of *x*.

30113 If the correct value would cause overflow, a range error shall occur and *exp*( ), *expf*( ), and *expl*( )
30114 shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

30115 MXX If the correct value would cause underflow, and is not representable, a range error may occur,
30116 MXX and *exp*( ), *expf*( ), and *expl*( ) shall return 0.0, or (if the IEC 60559 Floating-Point option is not
30117 supported) an implementation-defined value no greater in magnitude than DBL_MIN,
30118 FLT_MIN, and LDBL_MIN, respectively.

30119 MX If *x* is NaN, a NaN shall be returned.

30120 If *x* is ±0, 1 shall be returned.

30121 If *x* is –Inf, +0 shall be returned.

30122 If *x* is +Inf, *x* shall be returned.

30123 MXX If the correct value would cause underflow, and is representable, a range error may occur and
30124 the correct value shall be returned.

30125 **ERRORS**

30126 These functions shall fail if:

30127 Range Error The result overflows.

30128 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
30129 then *errno* shall be set to [ERANGE]. If the integer expression
30130 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
30131 floating-point exception shall be raised.

30132 These functions may fail if:

30133 Range Error The result underflows.

30134 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
30135 then *errno* shall be set to [ERANGE]. If the integer expression
30136 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
30137 floating-point exception shall be raised.

EXAMPLES

Computing the Density of the Standard Normal Distribution

This function shows an implementation for the density of the standard normal distribution using *exp*( ). This example uses the constant M_PI which is part of the XSI option.

```
#include <math.h>

double
normal_density (double x)
{
    return exp(-x*x/2) / sqrt (2*M_PI);
}
```

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( ), *log*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *expf*( ) and *expl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/26 is applied, adding the example to the EXAMPLES section.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0106 [68] and XSH/TC1-2008/0107 [68] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0101 [630] is applied.

NAME

exp2, exp2f, exp2l — exponential base 2 functions

SYNOPSIS

```
#include <math.h>

double exp2(double x);
float exp2f(float x);
long double exp2l(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the base-2 exponential of *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return 2^x.

If the correct value would cause overflow, a range error shall occur and *exp2*( ), *exp2f*( ), and *exp2l*( ) shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

MXX If the correct value would cause underflow, and is not representable, a range error may occur, and *exp2*( ), *exp2f*( ), and *exp2l*( ) shall return 0.0, or (if the IEC 60559 Floating-Point option is not supported) an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, 1 shall be returned.

If *x* is –Inf, +0 shall be returned.

If *x* is +Inf, *x* shall be returned.

MXX If the correct value would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

ERRORS

These functions shall fail if:

Range Error — The result overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

Range Error — The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow

floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exp( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *log*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0108 [68] and XSH/TC1-2008/0109 [68] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0102 [630] is applied.

30237 **NAME**

30238 expm1, expm1f, expm1l — compute exponential functions

30239 **SYNOPSIS**

30240
```
#include <math.h>
```

30241
```
double expm1(double x);
```
30242
```
float expm1f(float x);
```
30243
```
long double expm1l(long double x);
```

30244 **DESCRIPTION**

30245 CX The functionality described on this reference page is aligned with the ISO C standard. Any
30246 conflict between the requirements described here and the ISO C standard is unintentional. This
30247 volume of POSIX.1-2024 defers to the ISO C standard.

30248 These functions shall compute e^x–1.0.

30249 An application wishing to check for error situations should set *errno* to zero and call
30250 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
30251 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
30252 zero, an error has occurred.

30253 **RETURN VALUE**

30254 Upon successful completion, these functions return e^x–1.0.

30255 If the correct value would cause overflow, a range error shall occur and *expm1*( ), *expm1f*( ), and
30256 *expm1l*( ) shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
30257 respectively.

30258 MX If *x* is NaN, a NaN shall be returned.

30259 If *x* is ±0, ±0 shall be returned.

30260 If *x* is –Inf, –1 shall be returned.

30261 If *x* is +Inf, *x* shall be returned.

30262 MXX If *x* is subnormal, *x* should be returned.
30263 MX If *x* is subnormal, a range error may occur and, if *x* is not returned, *expm1*( ), *expm1f*( ), and
30264 *expm1l*( ) shall return an implementation-defined value no greater in magnitude than DBL_MIN,
30265 FLT_MIN, and LDBL_MIN, respectively.

30266 **ERRORS**

30267 These functions shall fail if:

30268 Range Error The result overflows.

30269 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
30270 then *errno* shall be set to [ERANGE]. If the integer expression
30271 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
30272 floating-point exception shall be raised.

30273 These functions may fail if:

30274 MX Range Error The value of *x* is subnormal.

30275 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
30276 then *errno* shall be set to [ERANGE]. If the integer expression
30277 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
30278 floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

The value of *expm1*(*x*) may be more accurate than *exp*(*x*)−1.0 for small values of *x*.

The *expm1*( ) and *log1p*( ) functions are useful for financial calculations of $((1+x)^n-1)/x$, namely:

```
expm1(n * log1p(x))/x
```

when *x* is very small (for example, when calculating small daily interest rates). These functions also simplify writing accurate inverse hyperbolic functions.

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exp( ), *feclearexcept*( ), *fetestexcept*( ), *ilogb*( ), *log1p*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The *expm1f*( ) and *expm1l*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The *expm1*( ) function is no longer marked as an extension.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0110 [68] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0103 [630] is applied.

Issue 8

Austin Group Defect 1382 is applied, rearranging the text describing the behavior when *x* is subnormal to avoid the need for two shading changes.

30314 **NAME**

30315 fabs, fabsf, fabsl — absolute value function

30316 **SYNOPSIS**

30317
```
#include <math.h>
```

30318
```
double fabs(double x);
```
30319
```
float fabsf(float x);
```
30320
```
long double fabsl(long double x);
```

30321 **DESCRIPTION**

30322 CX The functionality described on this reference page is aligned with the ISO C standard. Any
30323 conflict between the requirements described here and the ISO C standard is unintentional. This
30324 volume of POSIX.1-2024 defers to the ISO C standard.

30325 These functions shall compute the absolute value of their argument x, $|x|$.

30326 **RETURN VALUE**

30327 Upon successful completion, these functions shall return the absolute value of x.

30328 MX The returned value shall be exact and shall be independent of the current rounding direction
30329 mode.

30330 If x is NaN, a NaN shall be returned.

30331 If x is ±0, +0 shall be returned.

30332 If x is ±Inf, +Inf shall be returned.

30333 **ERRORS**

30334 No errors are defined.

30335 **EXAMPLES**

30336 **Computing the 1-Norm of a Floating-Point Vector**

30337 This example shows the use of *fabs*( ) to compute the 1-norm of a vector defined as follows:

30338
```
norm1(v) = |v[0]| + |v[1]| + ... + |v[n-1]|
```

30339 where $|x|$ denotes the absolute value of x, n denotes the vector's dimension $v[i]$ denotes the i-th
30340 component of v ($0 \le i < n$).

30341
```
#include <math.h>
```

30342
```
double
```
30343
```
norm1(const double v[], const int n)
```
30344
```
{
```
30345
```
    int     i;
```
30346
```
    double  n1_v;  /* 1-norm of v */
```

30347
```
    n1_v = 0;
```
30348
```
    for (i=0; i<n; i++) {
```
30349
```
        n1_v += fabs (v[i]);
```
30350
```
    }
```

30351
```
    return n1_v;
```
30352
```
}
```

30353 **APPLICATION USAGE**

30354 None.

30355 **RATIONALE**

30356 None.

30357 **FUTURE DIRECTIONS**

30358 None.

30359 **SEE ALSO**

30360 *isnan*( )

30361 XBD **<math.h>**

30362 **CHANGE HISTORY**

30363 First released in Issue 1. Derived from Issue 1 of the SVID.

30364 **Issue 5**

30365 The DESCRIPTION is updated to indicate how an application should check for an error. This
30366 text was previously published in the APPLICATION USAGE section.

30367 **Issue 6**

30368 The *fabsf*( ) and *fabsl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

30369 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
30370 revised to align with the ISO/IEC 9899:1999 standard.

30371 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
30372 marked.

30373 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/27 is applied, adding the example to the
30374 EXAMPLES section.

30375 **Issue 8**

30376 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018
30377 standard.

30378 **NAME**

30379 faccessat — determine accessibility of a file relative to directory file descriptor

30380 **SYNOPSIS**

30381 ```
#include <unistd.h>
```

30382 ```
int faccessat(int fd, const char *path, int amode, int flag);
```

30383 **DESCRIPTION**

30384 Refer to *access*( ).

30385 **NAME**

30386 fchdir — change working directory

30387 **SYNOPSIS**

30388 `#include <unistd.h>`

30389 `int fchdir(int fildes);`

30390 **DESCRIPTION**

30391 The *fchdir*( ) function shall be equivalent to *chdir*( ) except that the directory that is to be the new
30392 current working directory is specified by the file descriptor *fildes*.

30393 A conforming application can obtain a file descriptor for a file of type directory using *open*( ),
30394 provided that the file status flags and access modes do not contain O_WRONLY or O_RDWR.

30395 **RETURN VALUE**

30396 Upon successful completion, *fchdir*( ) shall return 0. Otherwise, it shall return –1 and set *errno* to
30397 indicate the error. On failure the current working directory shall remain unchanged.

30398 **ERRORS**

30399 The *fchdir*( ) function shall fail if:

30400 [EACCES] Search permission is denied for the directory referenced by *fildes*.

30401 [EBADF] The *fildes* argument is not an open file descriptor.

30402 [ENOTDIR] The open file descriptor *fildes* does not refer to a directory.

30403 The *fchdir*( ) may fail if:

30404 [EINTR] A signal was caught during the execution of *fchdir*( ).

30405 [EIO] An I/O error occurred while reading from or writing to the file system.

30406 **EXAMPLES**

30407 None.

30408 **APPLICATION USAGE**

30409 None.

30410 **RATIONALE**

30411 None.

30412 **FUTURE DIRECTIONS**

30413 None.

30414 **SEE ALSO**

30415 *chdir*( ), *dirfd*( )

30416 XBD **<unistd.h>**

30417 **CHANGE HISTORY**

30418 First released in Issue 4, Version 2.

30419 **Issue 5**

30420 Moved from X/OPEN UNIX extension to BASE.

30421 **Issue 7**

30422 The *fchdir*( ) function is moved from the XSI option to the Base.

30423 **NAME**

30424 fchmod — change mode of a file

30425 **SYNOPSIS**

30426
```
#include <sys/stat.h>
```

30427
```
int fchmod(int fildes, mode_t mode);
```

30428 **DESCRIPTION**

30429 The *fchmod*( ) function shall be equivalent to *chmod*( ) except that the file whose permissions are
30430 changed is specified by the file descriptor *fildes*.

30431 SHM If *fildes* references a shared memory object, the *fchmod*( ) function need only affect the S_IRUSR,
30432 S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.

30433 TYM If *fildes* references a typed memory object, the behavior of *fchmod*( ) is unspecified.

30434 If *fildes* refers to a socket, the behavior of *fchmod*( ) is unspecified.

30435 **RETURN VALUE**

30436 Upon successful completion, *fchmod*( ) shall return 0. Otherwise, it shall return –1 and set *errno* to
30437 indicate the error.

30438 **ERRORS**

30439 The *fchmod*( ) function shall fail if:

30440 [EBADF] The *fildes* argument is not an open file descriptor.

30441 [EPERM] The effective user ID does not match the owner of the file and the process does
30442 not have appropriate privileges.

30443 [EROFS] The file referred to by *fildes* resides on a read-only file system.

30444 The *fchmod*( ) function may fail if:

30445 XSI [EINTR] The *fchmod*( ) function was interrupted by a signal.

30446 XSI [EINVAL] The value of the *mode* argument is invalid.

30447 [EINVAL] The *fildes* argument refers to a pipe and the implementation disallows
30448 execution of *fchmod*( ) on a pipe.

30449 **EXAMPLES**

30450 **Changing the Current Permissions for a File**

30451 The following example shows how to change the permissions for a file named **/home/cnd/mod1**
30452 so that the owner and group have read/write/execute permissions, but the world only has
30453 read/write permissions.

30454
```
#include <sys/stat.h>
#include <fcntl.h>
```

30456
```
mode_t mode;
int    fildes;
...
fildes = open("/home/cnd/mod1", O_RDWR);
fchmod(fildes, S_IRWXU | S_IRWXG | S_IROTH | S_IWOTH);
```


APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *chown*( ), *creat*( ), *fcntl*( ), *fstatat*( ), *fstatvfs*( ), *mknod*( ), *open*( ), *read*( ), *write*( )

XBD **<sys/stat.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE and aligned with *fchmod*( ) in the POSIX Realtime Extension. Specifically, the second paragraph of the DESCRIPTION is added and a second instance of [EINVAL] is defined in the list of optional errors.

Issue 6

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by stating that *fchmod*( ) behavior is unspecified for typed memory objects.

Issue 8

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

30481 **NAME**

30482 fchmodat — change mode of a file relative to directory file descriptor

30483 **SYNOPSIS**

30484 ```
#include <sys/stat.h>
```

30485 ```
int fchmodat(int fd, const char *path, mode_t mode, int flag);
```

30486 **DESCRIPTION**

30487 Refer to *chmod( )*.

NAME

fchown — change owner and group of a file

SYNOPSIS

```
#include <unistd.h>

int fchown(int fildes, uid_t owner, gid_t group);
```

DESCRIPTION

The *fchown*( ) function shall be equivalent to *chown*( ) except that the file whose owner and group are changed is specified by the file descriptor *fildes*.

RETURN VALUE

Upon successful completion, *fchown*( ) shall return 0. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *fchown*( ) function shall fail if:

[EBADF] The *fildes* argument is not an open file descriptor.

[EPERM] The effective user ID does not match the owner of the file or the process does not have appropriate privileges and _POSIX_CHOWN_RESTRICTED indicates that such privilege is required.

[EROFS] The file referred to by *fildes* resides on a read-only file system.

The *fchown*( ) function may fail if:

[EINVAL] The owner or group ID is not a value supported by the implementation. The *fildes* argument refers to a pipe or socket and the implementation disallows execution of *fchown*( ) on a pipe.

[EIO] A physical I/O error has occurred.

[EINTR] The *fchown*( ) function was interrupted by a signal which was caught.

EXAMPLES

Changing the Current Owner of a File

The following example shows how to change the owner of a file named **/home/cnd/mod1** to ``jones'' and the group to ``cnd''.

The numeric value for the user ID is obtained by extracting the user ID from the user database entry associated with ``jones''. Similarly, the numeric value for the group ID is obtained by extracting the group ID from the group database entry associated with ``cnd''. This example assumes the calling program has appropriate privileges.

```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <pwd.h>
#include <grp.h>

struct passwd *pwd;
struct group  *grp;
int            fildes;
...
fildes = open("/home/cnd/mod1", O_RDWR);
pwd = getpwnam("jones");
```



```
        grp = getgrnam("cnd");
        fchown(fildes, pwd->pw_uid, grp->gr_gid);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

chown( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The following changes were made to align with the IEEE P1003.1a draft standard:

- Clarification is added that a call to *fchown*( ) may not be allowed on a pipe.

The *fchown*( ) function is defined as mandatory.

Issue 7

Functionality relating to XSI STREAMS is marked obsolescent.

Issue 8

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

30554 **NAME**

30555 fchownat — change owner and group of a file relative to directory file descriptor

30556 **SYNOPSIS**

30557 ```
#include <unistd.h>
```

30558 ```
int fchownat(int fd, const char *path, uid_t owner, gid_t group,
```

30559 ```
    int flag);
```

30560 **DESCRIPTION**

30561 Refer to *chown( )*.

  
```

30562 **NAME**

30563 fclose — close a stream

30564 **SYNOPSIS**

30565 ```
#include <stdio.h>
```

30566 ```
int fclose(FILE *stream);
```

30567 **DESCRIPTION**

30568 CX The functionality described on this reference page is aligned with the ISO C standard. Any
30569 conflict between the requirements described here and the ISO C standard is unintentional. This
30570 volume of POSIX.1-2024 defers to the ISO C standard.

30571 The *fclose*( ) function shall cause the stream pointed to by *stream* to be flushed and the associated
30572 file to be closed. Any unwritten buffered data for the stream shall be written to the file; any
30573 unread buffered data shall be discarded. Whether or not the call succeeds, the stream shall be
30574 disassociated from the file and any buffer set by the *setbuf*( ) or *setvbuf*( ) function shall be
30575 disassociated from the stream. If the associated buffer was automatically allocated, it shall be
30576 deallocated.

30577 CX If the file is not already at EOF, and the file is one capable of seeking, the file offset of the
30578 underlying open file description shall be set to the file position of the stream if the stream is the
30579 active handle to the underlying file description.

30580 The *fclose*( ) function shall mark for update the last data modification and last file status change
30581 timestamps of the underlying file, if the stream was writable, and if buffered data remains that
30582 has not yet been written to the file. The *fclose*( ) function shall perform the equivalent of a *close*( )
30583 on the file descriptor that is associated with the stream pointed to by *stream*.

30584 After the call to *fclose*( ), any use of *stream* results in undefined behavior.

30585 **RETURN VALUE**

30586 CX Upon successful completion, *fclose*( ) shall return 0; otherwise, it shall return EOF and set *errno*
30587 to indicate the error.

30588 **ERRORS**

30589 The *fclose*( ) function shall fail if:

30590 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and
30591 the thread would be delayed in the write operation.

30592 CX [EBADF] The file descriptor underlying stream is not valid.

30593 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

30594 CX [EFBIG] An attempt was made to write a file that exceeds the file size limit of the
30595 process.
30596 XSI A SIGXFSZ signal shall also be generated for the thread.

30597 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
30598 offset maximum associated with the corresponding stream.

30599 CX [EINTR] The *fclose*( ) function was interrupted by a signal.

30600 CX [EIO] The process is a member of a background process group attempting to write to
30601 its controlling terminal, TOSTOP is set, the calling thread is not blocking
30602 SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the
30603 process is orphaned. This error may also be returned under implementation-
30604 defined conditions.

CX [ENOMEM] The underlying stream was created by *open_memstream*( ) or *open_wmemstream*( ) and insufficient memory is available.

CX [ENOSPC] There was no free space remaining on the device containing the file or in the buffer used by the *fmemopen*( ) function.

CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal shall also be sent to the thread.

The *fclose*( ) function may fail if:

CX [ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.

EXAMPLES

None.

APPLICATION USAGE

Since after the call to *fclose*( ) any use of *stream* results in undefined behavior, *fclose*( ) should not be used on *stdin*, *stdout*, or *stderr* except immediately before process termination (see XBD Section 3.287, on page 73), so as to avoid triggering undefined behavior in other standard interfaces that rely on these streams. If there are any *atexit*( ) handlers registered by the application, such a call to *fclose*( ) should not occur until the last handler is finishing. Once *fclose*( ) has been used to close *stdin*, *stdout*, or *stderr*, there is no standard way to reopen any of these streams.

Use of *freopen*( ) to change *stdin*, *stdout*, or *stderr* instead of closing them avoids the danger of a file unexpectedly being opened as one of the special file descriptors STDIN_FILENO, STDOUT_FILENO, or STDERR_FILENO at a later time in the application.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *atexit*( ), *close*( ), *fmemopen*( ), *fopen*( ), *freopen*( ), *getrlimit*( ), *open_memstream*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EFBIG] error is added as part of the large file support extensions.
- The [ENXIO] optional error condition is added.

The DESCRIPTION is updated to note that the stream and any buffer are disassociated whether or not the call succeeds. This is for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/28 is applied, updating the [EAGAIN]

error in the ERRORS section from ``the process would be delayed'' to ``the thread would be delayed''.

Issue 7

Austin Group Interpretation 1003.1-2001 #002 is applied, clarifying the interaction of file descriptors and streams.

The [ENOSPC] error condition is updated and the [ENOMEM] error is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0113 [87], XSH/TC1-2008/0114 [79], and XSH/TC1-2008/0115 [14] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0104 [555] is applied.

Issue 8

Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error relating to the file size limit for the process.

30663 **NAME**

30664 fcntl — file control

30665 **SYNOPSIS**

30666 `#include <fcntl.h>`

30667 `int fcntl(int fildes, int cmd, ...);`

30668 **DESCRIPTION**

30669 The *fcntl*( ) function shall perform the operations described below on open files. The *fildes*
30670 argument is a file descriptor.

30671 The available values for *cmd* are defined in **<fcntl.h>** and are as follows:

30672 F_DUPFD Return a new file descriptor which shall be allocated as described in
30673 Section 2.6 (on page 525), except that it shall be the lowest numbered
30674 available file descriptor greater than or equal to the third argument, *arg*,
30675 taken as an integer of type **int**. The new file descriptor shall refer to the
30676 same open file description as the original file descriptor, and shall share
30677 any locks. The FD_CLOEXEC and FD_CLOFORK flags associated with
30678 the new file descriptor shall be cleared.

30679 F_DUPFD_CLOEXEC
30680 Like F_DUPFD, but the FD_CLOEXEC flag associated with the new file
30681 descriptor shall be set.

30682 F_DUPFD_CLOFORK
30683 Like F_DUPFD, but the FD_CLOFORK flag associated with the new file
30684 descriptor shall be set.

30685 F_GETFD Get the file descriptor flags defined in **<fcntl.h>** that are associated with
30686 the file descriptor *fildes*. File descriptor flags are associated with a single
30687 file descriptor and do not affect other file descriptors that refer to the
30688 same file.

30689 F_SETFD Set the file descriptor flags defined in **<fcntl.h>**, that are associated with
30690 *fildes*, to the third argument, *arg*, taken as type **int**. If the FD_CLOEXEC
30691 flag in the third argument is set, the file descriptor shall be closed upon
30692 SPN successful execution of an *exec* family function and in the new process
30693 image created by *posix_spawn*( ) or *posix_spawnp*( ); otherwise, the file
30694 descriptor shall remain open. If the FD_CLOFORK flag in the third
30695 argument is set, the file descriptor shall not be inherited by any child
30696 process created from a process that has the file descriptor open;
30697 otherwise, the file descriptor shall be inherited.

30698 F_GETFL Get the file status flags and file access modes, defined in **<fcntl.h>**, for the
30699 file description associated with *fildes*. The file access modes can be
30700 extracted from the return value using the mask O_ACCMODE, which is
30701 defined in **<fcntl.h>**. File status flags and file access modes are associated
30702 with the file description and do not affect other file descriptors that refer
30703 to the same file with different open file descriptions. The flags returned
30704 may include non-standard file status flags which the application did not
30705 set, provided that these additional flags do not alter the behavior of a
30706 conforming application.

30707 F_SETFL Set the file status flags, defined in **<fcntl.h>**, for the file description
30708 associated with *fildes* from the corresponding bits in the third argument,
30709 *arg*, taken as type **int**. Bits corresponding to the file access mode and the

30710 file creation flags, as defined in **<fcntl.h>**, that are set in *arg* shall be
30711 ignored. If any bits in *arg* other than those mentioned here are changed by
30712 the application, the result is unspecified. If *fildes* does not support non-
30713 blocking operations, it is unspecified whether the O_NONBLOCK flag
30714 will be ignored.

30715 F_GETOWN If *fildes* refers to a socket, get the process ID or process group ID specified
30716 to receive SIGURG signals when out-of-band data is available. Positive
30717 values shall indicate a process ID; negative values, other than –1, shall
30718 indicate a process group ID; the value zero shall indicate that no SIGURG
30719 signals are to be sent. If *fildes* does not refer to a socket, the results are
30720 unspecified.

30721 F_SETOWN If *fildes* refers to a socket, atomically set the process ID or process group
30722 ID specified to receive SIGURG signals when out-of-band data is
30723 available, using the value of the third argument, *arg*, taken as type **int**.
30724 Positive values shall indicate a process ID; negative values, other than –1,
30725 shall indicate a process group ID; the value zero shall indicate that no
30726 SIGURG signals are to be sent. If *fildes* does not refer to a socket, the
30727 results are unspecified.

30728 F_GETOWN_EX If *fildes* refers to a socket, get the process ID or process group ID specified
30729 to receive SIGURG signals when out-of-band data is available, by setting
30730 the *type* and *pid* members of the **f_owner_ex** structure pointed to by the
30731 third argument, *arg*. The value of *type* shall be F_OWNER_PID or
30732 F_OWNER_PGRP to indicate that *pid* contains a process ID or a process
30733 group ID, respectively. The value of *pid* shall be zero if no SIGURG signals
30734 are to be sent. If *fildes* does not refer to a socket, the results are
30735 unspecified.

30736 F_SETOWN_EX If *fildes* refers to a socket, set the process ID or process group ID specified
30737 to receive SIGURG signals when out-of-band data is available, using the
30738 value of the third argument, *arg*, taken as type pointer to **struct**
30739 **f_owner_ex**. The *type* and *pid* members of this structure shall be used as
30740 follows:

30741 • A *pid* value of zero shall indicate that no SIGURG signals are to be
30742 sent.

30743 • A *type* value of F_OWNER_PID and a positive *pid* value shall
30744 indicate that SIGURG signals are to be sent to the process ID
30745 specified in *pid*.

30746 • A *type* value of F_OWNER_PGRP and a positive *pid* value shall
30747 indicate that SIGURG signals are to be sent to the process group ID
30748 specified in *pid*.

30749 If *fildes* does not refer to a socket, the results are unspecified.

30750 For F_SETOWN and F_SETOWN_EX, each time a SIGURG signal is sent to the specified process
30751 or process group, permission checks equivalent to those performed by *kill*( ) shall be performed,
30752 as if *kill*( ) were called by a process with the same real user ID, effective user ID, and privileges
30753 that the process calling *fcntl*( ) has at the time of the call; if the *kill*( ) call would fail, no signal
30754 shall be sent. These permission checks may also be performed by the *fcntl*( ) call. If the process
30755 specified by *arg* later terminates, or the process group specified by *arg* later becomes empty,
30756 while still being specified to receive SIGURG signals when out-of-band data is available from

30757 *fildes*, then no signals shall be sent to any subsequently created process that has the same process
30758 ID or process group ID, regardless of permission; it is unspecified whether this is achieved by
30759 the equivalent of a *fcntl*(*fildes*, F_SETOWN, 0) call at the time the process terminates or is waited
30760 for or the process group becomes empty, or by other means.

30761 The following values for *cmd* are available for advisory record locking. Record locking shall be
30762 supported for regular files, and may be supported for other files.

30763 F_GETLK Get any lock which blocks the process-owned file lock description
30764 pointed to by the third argument, *arg*, taken as a pointer to type **struct**
30765 **flock**, defined in **<fcntl.h>**. The information retrieved shall overwrite the
30766 information passed to *fcntl*( ) in the structure **flock**. If no lock is found
30767 that would prevent this lock from being created, then the structure shall
30768 be left unchanged except for the lock type in *l_type* which shall be set to
30769 F_UNLCK.

30770 F_SETLK Set or clear a process-owned file lock according to the lock description
30771 pointed to by the third argument, *arg*, taken as a pointer to type **struct**
30772 **flock**, defined in **<fcntl.h>**. F_SETLK can establish shared (or read) locks
30773 (F_RDLCK) or exclusive (or write) locks (F_WRLCK), as well as remove
30774 either type of lock (F_UNLCK). F_RDLCK, F_WRLCK, and F_UNLCK are
30775 defined in **<fcntl.h>**. If a shared or exclusive lock cannot be set, *fcntl*( )
30776 shall return immediately with a return value of –1.

30777 F_SETLKW This command shall be equivalent to F_SETLK except that if a shared or
30778 exclusive lock is blocked by other locks, the thread shall wait until the
30779 request can be satisfied. If a signal that is to be caught is received while
30780 *fcntl*( ) is waiting for a region, *fcntl*( ) shall be interrupted. Upon return
30781 from the signal handler, *fcntl*( ) shall return –1 with *errno* set to [EINTR],
30782 and the lock operation shall not be done.

30783 F_OFD_GETLK Get any lock which blocks the OFD-owned file lock description pointed to
30784 by the third argument, *arg*, taken as a pointer to type **struct flock**, defined
30785 in **<fcntl.h>**; the application shall ensure that the *l_pid* member of the
30786 structure pointed to by *arg* is set to 0 on input. The information retrieved
30787 shall overwrite the information passed to *fcntl*( ) in the structure **flock**. If
30788 no lock is found that would prevent this lock from being created, then the
30789 structure shall be left unchanged except for the lock type in *l_type* which
30790 shall be set to F_UNLCK.

30791 F_OFD_SETLK Set or clear an OFD-owned file lock according to the lock description
30792 pointed to by the third argument, *arg*, taken as a pointer to type **struct**
30793 **flock**, defined in **<fcntl.h>**; the application shall ensure that the *l_pid*
30794 member of the structure pointed to by *arg* is set to 0 on input.
30795 F_OFD_SETLK can establish shared (or read) locks (F_RDLCK) or
30796 exclusive (or write) locks (F_WRLCK), as well as remove either type of
30797 lock (F_UNLCK). F_RDLCK, F_WRLCK, and F_UNLCK are defined in
30798 **<fcntl.h>**. If a shared or exclusive lock cannot be set, *fcntl*( ) shall return
30799 immediately with a return value of –1.

30800 F_OFD_SETLKW This command shall be equivalent to F_OFD_SETLK except that if a
30801 shared or exclusive lock is blocked by other locks, the thread shall wait
30802 until the request can be satisfied. If a signal that is to be caught is received
30803 while *fcntl*( ) is waiting for a region, *fcntl*( ) shall be interrupted. Upon
30804 return from the signal handler, *fcntl*( ) shall return –1 with *errno* set to

30805 [EINTR], and the lock operation shall not be done.

30806 Additional implementation-defined values for *cmd* may be defined in **<fcntl.h>**. Their names
30807 shall start with F_.

30808 When a shared lock is set on a segment of a file, other processes can set shared process-owned
30809 locks, and other open file descriptions can be used to set shared OFD-owned locks, on that
30810 segment or a portion of it. A shared process-owned lock shall prevent any other process from
30811 setting an exclusive process-owned lock, and shall prevent any exclusive OFD-owned lock from
30812 being set, on any portion of the protected area. A shared OFD-owned lock shall prevent any
30813 other open file description from being used to set an exclusive OFD-owned lock, and shall
30814 prevent any exclusive process-owned lock from being set, on any portion of the protected area.
30815 A request for a shared lock shall fail if the file descriptor is not open for reading.

30816 An exclusive process-owned lock shall prevent any other process from setting a shared or
30817 exclusive process-owned lock, and shall prevent any shared or exclusive OFD-owned lock from
30818 being set, on any portion of the protected area. An exclusive OFD-owned lock shall prevent any
30819 other open file description from being used to set a shared or exclusive OFD-owned lock, and
30820 shall prevent any shared or exclusive process-owned lock from being set, on any portion of the
30821 protected area. A request for an exclusive lock shall fail if the file descriptor is not open for
30822 writing.

30823 The structure **flock** describes the type (*l_type*), starting offset (*l_whence*), relative offset (*l_start*),
30824 size (*l_len*), and process ID (*l_pid*) of the segment of the file to be affected.

30825 The value of *l_whence* is SEEK_SET, SEEK_CUR, or SEEK_END, to indicate that the relative
30826 offset *l_start* bytes shall be measured from the start of the file, current position, or end of the file,
30827 respectively. The value of *l_len* is the number of consecutive bytes to be locked. The value of *l_len*
30828 may be negative (where the definition of **off_t** permits negative values of *l_len*). On input, the
30829 *l_pid* field shall be ignored for F_GETLK, F_SETLK and F_SETLKW; the application shall ensure
30830 that it is set to zero for F_OFD_GETLK, F_OFD_SETLK and F_OFD_SETLKW. It is set by
30831 F_GETLK and F_OFD_GETLK when identifying a blocking lock. After a successful F_GETLK or
30832 F_OFD_GETLK request, when a blocking lock is found, the values returned in the **flock**
30833 structure shall be as follows:

30834 *l_type* Type of blocking lock found.

30835 *l_whence* SEEK_SET.

30836 *l_start* Start of the blocking lock.

30837 *l_len* Length of the blocking lock.

30838 *l_pid* Process ID of the process that holds the blocking lock if the blocking lock is a
30839 process-owned file lock, or (**pid_t**)–1 if the blocking lock is an OFD-owned file
30840 lock.

30841 If the command is F_SETLKW or F_OFD_SETLKW and the thread needs to wait for a blocking
30842 lock to be released, then the range of bytes to be locked shall be determined before the *fcntl*( )
30843 function blocks. If the file size or file descriptor seek offset change while *fcntl*( ) is blocked, this
30844 shall not affect the range of bytes locked.

30845 If *l_len* is positive, the area affected shall start at *l_start* and end at *l_start*+*l_len*–1. If *l_len* is
30846 negative, the area affected shall start at *l_start*+*l_len* and end at *l_start*–1. Locks may start and
30847 extend beyond the current end of a file, but shall not extend before the beginning of the file. A
30848 lock shall be set to extend to the largest possible value of the file offset for that file by setting
30849 *l_len* to 0. If such a lock also has *l_start* set to 0 and *l_whence* is set to SEEK_SET, the whole file
30850 shall be locked.

Each byte in the file can be locked either with one or more shared locks (F_RDLCK) or with one exclusive lock (F_WRLCK).

Before a successful return from an F_SETLK or an F_SETLKW request when the calling process has previously existing process-owned locks on bytes in the region specified by the request, the previous shared or exclusive lock for each byte in the specified region shall be replaced by the new shared or exclusive lock. An F_SETLK or an F_SETLKW request (respectively) shall fail or block when another process has existing process-owned locks, or any open file description (including the one associated with *fildes*) has existing OFD-owned locks, on bytes in the specified region and any of those locks conflicts with the requested lock.

Before a successful return from an F_OFD_SETLK or an F_OFD_SETLKW request when the open file description associated with fildes has previously existing OFD-owned locks on bytes in the region specified by the request, the previous shared or exclusive lock for each byte in the specified region shall be replaced by the new shared or exclusive lock. An F_OFD_SETLK or an F_OFD_SETLKW request (respectively) shall fail or block when another open file description has existing OFD-owned locks, or any process (including the calling process) has existing process-owned locks, on bytes in the specified region and any of those locks conflicts with the requested lock.

All process-owned locks associated with a file for a given process shall be removed when any file descriptor for that file is closed by that process (even if via a different open file description) or the process holding that file descriptor terminates. Process-owned locks shall not be inherited by a child process.

All OFD-owned locks associated with a given open file description shall be removed when all file descriptors associated with that open file description have been closed (either directly or as a side-effect of, for example, process termination or FD_CLOEXEC). OFD-owned locks shall be shared across all file descriptors that are associated with the owning open file description, regardless of which process holds the file descriptor.

A potential for deadlock occurs if a process or thread controlling a locked region is put to sleep by attempting to lock a region that has an existing conflicting lock. If the system detects that sleeping until a locked region is unlocked would cause a deadlock, *fcntl*( ) shall fail with an [EDEADLK] error. Deadlock detection may differ between process-owned locks and OFD-owned locks.

XSI The interaction between *fcntl*( ) and *lockf*( ) locks is unspecified.

An unlock (F_UNLCK) request in which *l_len* is non-zero and the offset of the last byte of the requested segment is the maximum value for an object of type **off_t**, when the process (for F_SETLK and F_SETLKW) or open file description (for F_OFD_SETLK and F_OFD_SETLKW) has an existing lock in which *l_len* is 0 and which includes the last byte of the requested segment, shall be treated as a request to unlock from the start of the requested segment with an *l_len* equal to 0. Otherwise, an unlock (F_UNLCK) request shall attempt to unlock only the requested segment.

SHM When the file descriptor *fildes* refers to a shared memory object, the behavior of *fcntl*( ) shall be the same as for a regular file except the effect of the following values for the argument *cmd* is unspecified: F_SETFL, F_GETLK, F_SETLK, F_SETLKW, F_OFD_GETLK, F_OFD_SETLK, and F_OFD_SETLKW.

TYM If *fildes* refers to a typed memory object, the result of the *fcntl*( ) function is unspecified.

RETURN VALUE

Upon successful completion, the value returned shall depend on *cmd* as follows:

F_DUPFD — A new file descriptor.

F_DUPFD_CLOEXEC — A new file descriptor.

F_DUPFD_CLOFORK — A new file descriptor.

F_GETFD — Value of flags defined in **<fcntl.h>**. The return value shall not be negative.

F_SETFD — Value other than –1.

F_GETFL — Value of file status flags and access modes. The return value shall not be negative.

F_SETFL — Value other than –1.

F_GETLK — Value other than –1.

F_SETLK — Value other than –1.

F_SETLKW — Value other than –1.

F_OFD_GETLK — Value other than –1.

F_OFD_SETLK — Value other than –1.

F_OFD_SETLKW — Value other than –1.

F_GETOWN — Value of the socket owner process or process group; this shall not be –1.

F_SETOWN — Value other than –1.

F_GETOWN_EX — Value other than –1.

F_SETOWN_EX — Value other than –1.

Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *fcntl*( ) function shall fail if:

[EACCES] or [EAGAIN]
The *cmd* argument is F_SETLK, the type of lock (*l_type*) is a shared (F_RDLCK) or exclusive (F_WRLCK) lock, and the requested lock cannot be set because it is blocked by an existing lock on the file.

[EAGAIN] The *cmd* argument is F_OFD_SETLK, the type of lock (*l_type*) is a shared (F_RDLCK) or exclusive (F_WRLCK) lock, and the requested lock cannot be set because it is blocked by an existing lock on the file.

[EBADF] The *fildes* argument is not a valid open file descriptor; or the argument *cmd* is F_SETLK, F_SETLKW, F_OFD_SETLK, or F_OFD_SETLKW, the type of lock, *l_type*, is a shared lock (F_RDLCK), and *fildes* is not a valid file descriptor open for reading, or the type of lock, *l_type*, is an exclusive lock (F_WRLCK), and *fildes* is not a valid file descriptor open for writing.

30933 [EINTR] The *cmd* argument is F_SETLKW or F_OFD_SETLKW and the function was
30934 interrupted by a signal.

30935 [EINVAL] The *cmd* argument is invalid; or the *cmd* argument is F_DUPFD,
30936 F_DUPFD_CLOEXEC, or F_DUPFD_CLOFORK and *arg* is negative or is
30937 greater than or equal to {OPEN_MAX}; or the *cmd* argument is
30938 F_SETOWN_EX and the *type* member of the **f_owner_ex** structure pointed to
30939 by *arg* is invalid, or the *pid* member is negative and the *type* member is
30940 F_OWNER_PID or F_OWNER_PGRP; or the *cmd* argument is F_GETLK,
30941 F_SETLK, F_SETLKW, F_OFD_GETLK, F_OFD_SETLK, or F_OFD_SETLKW
30942 and the data pointed to by *arg* is not valid, or *fildes* refers to a file that does not
30943 support locking.

30944 [EMFILE] The argument *cmd* is F_DUPFD, F_DUPFD_CLOEXEC, or
30945 F_DUPFD_CLOFORK and all file descriptors available to the process are
30946 currently open, or no file descriptors greater than or equal to *arg* are available.

30947 [ENOLCK] The argument *cmd* is F_SETLK, F_SETLKW, F_OFD_SETLK, or
30948 F_OFD_SETLKW and satisfying the lock or unlock request would result in the
30949 number of locked regions in the system exceeding a system-imposed limit.

30950 [EOVERFLOW] One of the values to be returned cannot be represented correctly.

30951 [EOVERFLOW] The *cmd* argument is F_GETLK, F_SETLK, F_SETLKW, F_OFD_GETLK,
30952 F_OFD_SETLK, or F_OFD_SETLKW and the smallest or, if *l_len* is non-zero,
30953 the largest offset of any byte in the requested segment cannot be represented
30954 correctly in an object of type **off_t**.

30955 [ESRCH] The *cmd* argument is F_SETOWN or F_SETOWN_EX and no process or
30956 process group can be found corresponding to that specified by *arg*.

30957 The *fcntl*( ) function may fail if:

30958 [EDEADLK] The *cmd* argument is F_SETLKW or F_OFD_SETLKW, the type of lock (*l_type*)
30959 is a shared (F_RDLCK) or exclusive (F_WRLCK) lock, the requested lock is
30960 blocked by an existing lock on the file, and the system determines that waiting
30961 for that lock to be released would cause a deadlock.

30962 [EINVAL] The *cmd* argument is F_SETOWN and the value of *arg* is positive and is not
30963 valid as a process ID or the value of *arg* is negative and its absolute value is
30964 not valid as a process group ID; or the *cmd* argument is F_SETOWN_EX, the
30965 value of the *type* member of the **f_owner_ex** structure pointed to by *arg* is
30966 F_OWNER_PID, and the value of the *pid* member is not valid as a process ID;
30967 or the *cmd* argument is F_SETOWN_EX, the value of the *type* member of the
30968 **f_owner_ex** structure pointed to by *arg* is F_OWNER_PGRP, and the value of
30969 the *pid* member is not valid as a process group ID.

30970 [EPERM] The *cmd* argument is F_SETOWN or F_SETOWN_EX and the calling process
30971 does not have permission to send a SIGURG signal to any process specified by
30972 *arg*.

EXAMPLES

Locking and Unlocking a File

The following example demonstrates how to place a lock on bytes 100 to 109 of a file and then later remove it. F_SETLK is used to perform a non-blocking lock request so that the process does not have to wait if an incompatible lock is held by another process; instead the process can take some other action.

```
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>

int
main(int argc, char *argv[])
{
    int fd;
    struct flock fl;

    fd = open("testfile", O_RDWR);
    if (fd == -1)
        /* Handle error */;

    /* Make a non-blocking request to place a write lock
       on bytes 100-109 of testfile */

    fl.l_type = F_WRLCK;
    fl.l_whence = SEEK_SET;
    fl.l_start = 100;
    fl.l_len = 10;

    if (fcntl(fd, F_SETLK, &fl) == -1) {
        if (errno == EACCES || errno == EAGAIN) {
            printf("Already locked by another process\n");

            /* We cannot get the lock at the moment */

        } else {
            /* Handle unexpected error */;
        }
    } else { /* Lock was granted... */

        /* Perform I/O on bytes 100 to 109 of file */

        /* Unlock the locked bytes */

        fl.l_type = F_UNLCK;
        fl.l_whence = SEEK_SET;
        fl.l_start = 100;
        fl.l_len = 10;
        if (fcntl(fd, F_SETLK, &fl) == -1)
            /* Handle error */;
    }
    exit(EXIT_SUCCESS);
} /* main */
```


Setting the Close-on-Exec Flag

The following example demonstrates how to set the close-on-exec flag for the file descriptor *fd*.

```
#include <unistd.h>
#include <fcntl.h>
...
    int flags;

    flags = fcntl(fd, F_GETFD);
    if (flags == -1)
        /* Handle error */;
    flags |= FD_CLOEXEC;
    if (fcntl(fd, F_SETFD, flags) == -1)
        /* Handle error */;"
```

APPLICATION USAGE

The *arg* values to F_GETFD, F_SETFD, F_GETFL, and F_SETFL all represent flag values to allow for future growth. Applications using these functions should do a read-modify-write operation on them, rather than assuming that only the values defined by this volume of POSIX.1-2024 are valid. It is a common error to forget this, particularly in the case of F_SETFD. Some implementations set additional file status flags to advise the application of default behavior, even though the application did not request these flags.

In order to set both FD_CLOEXEC and FD_CLOFORK when duplicating a file descriptor, applications should use F_DUPFD_CLOFORK to obtain the new file descriptor with FD_CLOFORK already set, and then use F_SETFD to set the FD_CLOEXEC flag on the new descriptor. (The alternative of first using F_DUPFD_CLOEXEC and then setting FD_CLOFORK with F_SETFD has a timing window where another thread could create a child process which inherits the new descriptor because FD_CLOFORK has not yet been set.)

The FD_CLOFORK flag takes effect for all child processes, not just those created using *fork*( ) or *_Fork*( ).

On implementations where process IDs can be greater than {INT_MAX}, F_SETOWN cannot be used with process IDs greater than {INT_MAX} or process group IDs greater than {INT_MAX}+1 because the value is passed to *fcntl*( ) in an argument of type **int**. In this situation, F_SETOWN_EX should be used instead.

Similarly, if a process ID greater than {INT_MAX} or a process group ID greater than {INT_MAX}+1 has been set to receive SIGURG signals (using F_SETOWN_EX), F_GETOWN cannot be used to obtain the value because *fcntl*( ) returns the value as type **int** and will thus give an [EOVERFLOW] error for such values. F_GETOWN_EX should be used instead.

Note that the convention of negating a process group ID is only used with F_SETOWN and F_GETOWN; the *pid* member of the **f_owner_ex** structure used with F_SETOWN_EX and F_GETOWN_EX is not negated when it specifies a process group ID.

On systems which do not perform permission checks at the time of an *fcntl*( ) call with F_SETOWN or F_SETOWN_EX, if the permission checks performed at the time the signal is sent disallow sending the signal to any process, the process that called *fcntl*( ) has no way of discovering that this has happened. A call to *kill*( ) with signal 0 can be used as a prior check of permissions, although this is no guarantee that permission will be granted at the time a signal is sent, since the target process(es) could change user IDs or privileges in the meantime.

Record-locking should not be used in combination with buffered standard I/O streams (see Section 2.5, on page 521). Instead, non-buffered I/O should be used. Unexpected results may

occur in processes that do buffering in the user address space. The process may later read/write data which is/was locked. Functions that operate on standard I/O streams are the most common source of such buffering.

RATIONALE

The ellipsis in the SYNOPSIS is the syntax specified by the ISO C standard for a variable number of arguments. It is used because System V uses pointers for the implementation of file locking functions.

This volume of POSIX.1-2024 permits concurrent read and write access to file data using the *fcntl*( ) function; this is a change from the 1984 /usr/group standard and early proposals. Without concurrency controls, this feature may not be fully utilized without occasional loss of data.

Data losses occur in several ways. One case occurs when several processes try to update the same record, without sequencing controls; several updates may occur in parallel and the last writer ``wins''. Another case is a bit-tree or other internal list-based database that is undergoing reorganization. Without exclusive use to the tree segment by the updating process, other reading processes chance getting lost in the database when the index blocks are split, condensed, inserted, or deleted. While *fcntl*( ) is useful for many applications, it is not intended to be overly general and does not handle the bit-tree example well.

This facility is only required for regular files because it is not appropriate for many devices such as terminals and network connections.

Since *fcntl*( ) works with ``any file descriptor associated with that file, however it is obtained'', the file descriptor may have been inherited through a *fork*( ) or *exec* operation and thus may affect a file that another process also has open.

The use of the open file description to identify what to lock requires extra calls and presents problems if several processes are sharing an open file description, but there are too many implementations of the existing mechanism for this volume of POSIX.1-2024 to use different specifications.

Another consequence of this model is that closing any file descriptor for a given file (whether or not it is the same open file description that created the lock) causes the locks on that file to be relinquished for that process. Equivalently, any close for any file/process pair relinquishes the locks owned on that file for that process. But note that while an open file description may be shared through *fork*( ), locks are not inherited through *fork*( ). Yet locks may be inherited through one of the *exec* functions.

The identification of a machine in a network environment is outside the scope of this volume of POSIX.1-2024. Thus, an *l_sysid* member, such as found in System V, is not included in the locking structure.

Changing of lock types can result in a previously locked region being split into smaller regions.

Mandatory locking was a major feature of the 1984 /usr/group standard.

For advisory file record locking to be effective, all processes that have access to a file must cooperate and use the advisory mechanism before doing I/O on the file. Enforcement-mode record locking is important when it cannot be assumed that all processes are cooperating. For example, if one user uses an editor to update a file at the same time that a second user executes another process that updates the same file and if only one of the two processes is using advisory locking, the processes are not cooperating. Enforcement-mode record locking would protect against accidental collisions.

Secondly, advisory record locking requires a process using locking to bracket each I/O operation

with lock (or test) and unlock operations. With enforcement-mode file and record locking, a process can lock the file once and unlock when all I/O operations have been completed. Enforcement-mode record locking provides a base that can be enhanced; for example, with sharable locks. That is, the mechanism could be enhanced to allow a process to lock a file so other processes could read it, but none of them could write it.

Mandatory locks were omitted for several reasons:

1. Mandatory lock setting was done by multiplexing the set-group-ID bit in most implementations; this was confusing, at best.
2. The relationship to file truncation as supported in 4.2 BSD was not well specified.
3. Any publicly readable file could be locked by anyone. Many historical implementations keep the password database in a publicly readable file. A malicious user could thus prohibit logins. Another possibility would be to hold open a long-distance telephone line.
4. Some demand-paged historical implementations offer memory mapped files, and enforcement cannot be done on that type of file.

Since sleeping on a region is interrupted with any signal, *alarm*( ) may be used to provide a timeout facility in applications requiring it. This is useful in deadlock detection. Since implementation of full deadlock detection is not always feasible, the [EDEADLK] error was made optional.

The F_SETOWN_EX and F_GETOWN_EX values for *cmd* and the associated **f_owner_ex** structure were adopted from the GNU C library. In addition to the values F_OWNER_PID and F_OWNER_PGRP for the *type* member, this also has F_OWNER_TID to specify that the *pid* member contains a thread ID. However, this relies on thread IDs being representable in a **pid_t** and so was not included in POSIX.1-2024. The aim of adding F_SETOWN_EX and F_GETOWN_EX was to address the inability of F_SETOWN and F_GETOWN to handle process IDs greater than {INT_MAX} and process group IDs greater than {INT_MAX}+1, and this need is satisfied without including F_OWNER_TID.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *close*( ), *exec*, *kill*( ), *open*( ), *sigaction*( )

XBD **<fcntl.h>**, **<signal.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Large File Summit extensions are added.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

- In the DESCRIPTION, sentences describing behavior when *l_len* is negative are now mandated, and the description of unlock (F_UNLOCK) when *l_len* is non-negative is mandated.
- In the ERRORS section, the [EINVAL] error condition has the case mandated when the *cmd* is invalid, and two [EOVERFLOW] error conditions are added.

The F_GETOWN and F_SETOWN values are added for sockets.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Clarification is added that the extent of the bytes locked is determined prior to the blocking action.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *fcntl*( ) results are unspecified for typed memory objects.

The normative text is updated to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/29 is applied, adding the example to the EXAMPLES section.

Issue 7

Austin Group Interpretation 1003.1-2001 #150 is applied, clarifying the file status flags returned when *cmd* is F_GETFL.

Austin Group Interpretation 1003.1-2001 #171 is applied, adding support to set the FD_CLOEXEC flag atomically at *open*( ), and adding the F_DUPFD_CLOEXEC flag.

The optional **<unistd.h>** header is removed from this function, since **<fcntl.h>** now defines SEEK_SET, SEEK_CUR, and SEEK_END as part of the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0116 [141] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0105 [835], XSH/TC2-2008/0106 [677], XSH/TC2-2008/0107 [484], XSH/TC2-2008/0108 [675], and XSH/TC2-2008/0109 [675,677] are applied.

Issue 8

Austin Group Defect 695 is applied, adding an atomicity requirement to the F_SETOWN operation.

Austin Group Defects 768 and 1671 are applied, adding OFD-owned file locks.

Austin Group Defect 1203 is applied, changing some wording in the RETURN VALUE section to use ``shall''.

Austin Group Defects 1274 and 1670 are applied, adding F_GETOWN_EX and F_SETOWN_EX.

Austin Group Defect 1318 is applied, adding FD_CLOFORK and F_DUPFD_CLOFORK.

NAME

fdatasync — synchronize the data of a file (**REALTIME**)

SYNOPSIS

SIO
```
#include <unistd.h>

int fdatasync(int fildes);
```

DESCRIPTION

The *fdatasync*( ) function shall force all currently queued I/O operations associated with the file indicated by file descriptor *fildes* to the synchronized I/O completion state.

The functionality shall be equivalent to *fsync*( ) with the symbol _POSIX_SYNCHRONIZED_IO defined, with the exception that all I/O operations shall be completed as defined for synchronized I/O data integrity completion.

RETURN VALUE

If successful, the *fdatasync*( ) function shall return the value 0; otherwise, the function shall return the value –1 and set *errno* to indicate the error. If the *fdatasync*( ) function fails, outstanding I/O operations are not guaranteed to have been completed.

ERRORS

The *fdatasync*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

[EINVAL] This implementation does not support synchronized I/O for this file.

In the event that any of the queued I/O operations fail, *fdatasync*( ) shall return the error conditions defined for *read*( ) and *write*( ).

EXAMPLES

None.

APPLICATION USAGE

Note that even if the file descriptor is not open for writing, if there are any pending write requests on the underlying file, then that I/O will be completed prior to the return of *fdatasync*( ).

An application that modifies a directory, for example, by creating a file in the directory, can invoke *fdatasync*( ) on the directory to ensure that the directory's entries are synchronized, although for most applications this should not be necessary (see XBD Section 4.11, on page 98).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_fsync( ), *fcntl*( ), *fsync*( ), *open*( ), *read*( ), *write*( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Synchronized Input and Output option.

The *fdatasync*( ) function is marked as part of the Synchronized Input and Output option.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0110 [501] is applied.

Issue 8

Austin Group Defect 672 is applied, changing the APPLICATION USAGE section.

NAME

fdim, fdimf, fdiml — compute positive difference between two floating-point numbers

SYNOPSIS

```
#include <math.h>

double fdim(double x, double y);
float fdimf(float x, float y);
long double fdiml(long double x, long double y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall determine the positive difference between their arguments. If *x* is greater than *y*, *x*–*y* is returned. If *x* is less than or equal to *y*, +0 is returned.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the positive difference value.

If *x*–*y* is positive and overflows, a range error shall occur and *fdim*( ), *fdimf*( ), and *fdiml*( ) shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

If the correct value would cause underflow, a range error may occur, and *fdim*( ), *fdimf*( ), and MXX *fdiml*( ) shall return the correct value, or (if the IEC 60559 Floating-Point option is not supported) an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

MX If *x* or *y* is NaN, a NaN shall be returned.

ERRORS

The *fdim*( ) function shall fail if:

Range Error — The result overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

The *fdim*( ) function may fail if:

Range Error — The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *fmax*( ), *fmin*( )

Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0119 [68,428] and XSH/TC1-2008/0120 [68,428] are applied.

NAME

fdopen — associate a stream with a file descriptor

SYNOPSIS

CX
```
#include <stdio.h>

FILE *fdopen(int fildes, const char *mode);
```

DESCRIPTION

The *fdopen*( ) function shall associate a stream with a file descriptor.

The *mode* argument points to a character string that is valid for *fopen*( ). If the string begins with one of the following characters, then the stream shall be associated with *fildes* as specified. Otherwise, the behavior is undefined.

`'r'` If *mode* includes `'+'`, the associated stream shall be open for update (reading and writing); otherwise, the stream shall be open for reading only. If the open file description referenced by *fildes* has O_APPEND set, it shall remain set.

`'w'` If *mode* includes `'+'`, the associated stream shall be open for update (reading and writing); otherwise, the stream shall be open for writing only. The file shall not be truncated by the *fdopen*( ) call. If the open file description referenced by *fildes* has O_APPEND set, it shall remain set.

`'a'` If *mode* includes `'+'`, the associated stream shall be open for update (reading and writing); otherwise, the stream shall be open for writing only. If the open file description referenced by *fildes* has O_APPEND clear, it is unspecified whether O_APPEND is set by the *fdopen*( ) call or remains clear.

The presence of `'x'` in *mode* shall have no effect. The FD_CLOEXEC flag of *fildes* shall be unchanged if `'e'` is not present, and shall be set by the *fdopen*( ) call if `'e'` is present.

Additional values for the *mode* argument may be supported by an implementation.

The application shall ensure that the mode of the stream as expressed by the *mode* argument is allowed by the file access mode of the open file description to which *fildes* refers. The file position indicator associated with the new stream is set to the position indicated by the file offset associated with the file descriptor.

The error and end-of-file indicators for the stream shall be cleared. The *fdopen*( ) function may cause the last data access timestamp of the underlying file to be marked for update.

SHM If *fildes* refers to a shared memory object, the result of the *fdopen*( ) function is unspecified.

TYM If *fildes* refers to a typed memory object, the result of the *fdopen*( ) function is unspecified.

The *fdopen*( ) function shall preserve the offset maximum previously set for the open file description corresponding to *fildes*.

RETURN VALUE

Upon successful completion, *fdopen*( ) shall return a pointer to a stream; otherwise, a null pointer shall be returned and *errno* set to indicate the error.

ERRORS

The *fdopen*( ) function shall fail if:

[EMFILE] {STREAM_MAX} streams are currently open in the calling process.

The *fdopen*( ) function may fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

[EINVAL] The *mode* argument is not a valid mode.

[EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

[ENOMEM] Insufficient space to allocate a buffer.

EXAMPLES

None.

APPLICATION USAGE

File descriptors are obtained from calls like *open*( ), *dup*( ), *creat*( ), or *pipe*( ), which open files but do not return streams.

RATIONALE

The file descriptor may have been obtained from *open*( ), *creat*( ), *pipe*( ), *dup*( ), *fcntl*( ), or *socket*( ); inherited through *fork*( ), *posix_spawn*( ), or *exec*; or perhaps obtained by other means.

The meanings of the *mode* arguments of *fdopen*( ) and *fopen*( ) differ. With *fdopen*( ), write (`'w'`) mode cannot create or truncate a file, and append (`'a'`) mode cannot create a file. Inclusion of a `'b'` in the *mode* argument is allowed for consistency with *fopen*( ); the `'b'` has no effect on the resulting stream. Implementations differ as to whether specifying append (`'a'`) mode causes the O_APPEND flag to be set if it was clear, but they are encouraged to do so. Since *fdopen*( ) does not create a file, the `'x'` mode modifier is silently ignored. The `'e'` mode modifier is not strictly necessary for *fdopen*( ), since FD_CLOEXEC must not be changed when it is absent; however, it is standardized here since that modifier is necessary to avoid a data race in multi-threaded applications using *freopen*( ), and consistency dictates that all functions accepting *mode* strings should allow the same set of strings.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5.1 (on page 522), *fclose*( ), *fmemopen*( ), *fopen*( ), *open*( ), *open_memstream*( ), *posix_spawn*( ), *socket*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

Large File Summit extensions are added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the use and setting of the *mode* argument are changed to include binary streams.
- In the DESCRIPTION, text is added for large file support to indicate setting of the offset maximum in the open file description.
- All errors identified in the ERRORS section are added.

- In the DESCRIPTION, text is added that the *fdopen*( ) function may cause *st_atime* to be updated.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Clarification is added that it is the responsibility of the application to ensure that the mode is compatible with the open file descriptor.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *fdopen*( ) results are unspecified for typed memory objects.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/30 is applied, making corrections to the RATIONALE.

Issue 7

SD5-XSH-ERN-149 is applied, adding the {STREAM_MAX} [EMFILE] error condition.

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0121 [409] is applied.

Issue 8

Austin Group Defects 411 and 1526 are applied, changing the requirements for the *mode* argument.

NAME

fdopendir, opendir — open directory associated with file descriptor

SYNOPSIS

```
#include <dirent.h>

DIR *fdopendir(int fd);
DIR *opendir(const char *dirname);
```

DESCRIPTION

The *fdopendir*( ) function shall be equivalent to the *opendir*( ) function except that the directory is specified by a file descriptor rather than by a name. The file offset associated with the file descriptor at the time of the call determines which entries are returned.

Upon successful return from *fdopendir*( ), the file descriptor is under the control of the system, and if any attempt is made to close the file descriptor, or to modify the state of the associated description, other than by means of *closedir*( ), *readdir*( ), *readdir_r*( ), *rewinddir*( ), or *seekdir*( ), the behavior is undefined. Upon calling *closedir*( ) the file descriptor shall be closed.

XSI

It is unspecified whether the FD_CLOEXEC flag will be set on the file descriptor by a successful call to *fdopendir*( ) if it was not previously set. However, the flag shall not be cleared if it was previously set.

The *opendir*( ) function shall open a directory stream corresponding to the directory named by the *dirname* argument. The directory stream shall be positioned at the first entry. If *opendir*( ) opens a file descriptor for *dirname* to associate with the returned stream:

- The descriptor shall be allocated as if the O_DIRECTORY and O_CLOEXEC flags were passed to *open*( ).
- The descriptor shall be subject to the limit of {OPEN_MAX} file descriptors available to the process.

RETURN VALUE

Upon successful completion, these functions shall return a pointer to an object of type **DIR**. Otherwise, these functions shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *fdopendir*( ) function shall fail if:

[EBADF] The *fd* argument is not a valid file descriptor open for reading.

[ENOTDIR] The descriptor *fd* is not associated with a directory.

The *opendir*( ) function shall fail if:

[EACCES] Search permission is denied for the component of the path prefix of *dirname* or read permission is denied for *dirname*.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *dirname* argument.

[ENAMETOOLONG] The length of a component of a pathname is longer than {NAME_MAX}.

[ENOENT] A component of *dirname* does not name an existing directory or *dirname* is an empty string.

[ENOTDIR] A component of *dirname* names an existing file that is neither a directory nor a symbolic link to a directory.

The *opendir*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *dirname* argument.

[EMFILE] All file descriptors available to the process are currently open.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

[ENFILE] Too many files are currently open in the system.

EXAMPLES

Open a Directory Stream

The following program fragment demonstrates how the *opendir*( ) function is used.

```
#include <dirent.h>
...
    DIR *dir;
    struct dirent *dp;
...
    if ((dir = opendir (".")) == NULL) {
        perror ("Cannot open .");
        exit (1);
    }

    while ((dp = readdir (dir)) != NULL) {
...
```

Find And Open a File

The following program searches through a given directory looking for files whose name does not begin with a dot and whose size is larger than 1 MiB.

```
#include <stdio.h>
#include <dirent.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdint.h>
#include <stdlib.h>
#include <unistd.h>

int
main(int argc, char *argv[])
{
    struct stat statbuf;
    DIR *d;
    struct dirent *dp;
    int dfd, ffd;

    if ((d = fdopendir((dfd = open("./tmp", O_RDONLY)))) == NULL) {
        fprintf(stderr, "Cannot open ./tmp directory\n");
        exit(1);
    }
    while ((dp = readdir(d)) != NULL) {
```



```
            if (dp->d_name[0] == '.')
                continue;
            /* there is a possible race condition here as the file
             * could be renamed between the readdir and the open */
            if ((ffd = openat(dfd, dp->d_name, O_RDONLY)) == -1) {
                perror(dp->d_name);
                continue;
            }
            if (fstat(ffd, &statbuf) == 0 && statbuf.st_size > (1024*1024)) {
                /* found it ... */
                printf("%s: %jdK\n", dp->d_name,
                    (intmax_t)(statbuf.st_size / 1024));
            }
            close(ffd);
        }
        closedir(d); // note this implicitly closes dfd
        return 0;
    }
```

APPLICATION USAGE

The *opendir*( ) function should be used in conjunction with *readdir*( ), *closedir*( ), and *rewinddir*( ) to examine the contents of the directory (see the EXAMPLES section in *readdir*( )). This method is recommended for portability.

RATIONALE

The purpose of the *fdopendir*( ) function is to enable opening files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *opendir*( ), resulting in unspecified behavior.

Based on historical implementations, the rules about file descriptors apply to directory streams as well. However, this volume of POSIX.1-2024 does not mandate that *opendir*( ) opens a file descriptor to associate with the stream; this may instead be done by the first call to *dirfd*( ), thus avoiding the need to allocate a file descriptor if *dirfd*( ) is never called. Once a file descriptor has been associated with the stream, it is mandatory that *closedir*( ) deallocate the file descriptor. If *opendir*( ) opens a file descriptor to associate with the stream, it behaves as if the O_CLOEXEC flag for *open*( ) had been used, so that the FD_CLOEXEC flag is set for the file descriptor. If *fdopendir*( ) is used to create a directory stream, it is unspecified whether the FD_CLOEXEC flag on the file descriptor specified by the *fd* argument is set or left unchanged.

The directory entries for dot and dot-dot are optional. This volume of POSIX.1-2024 does not provide a way to test *a priori* for their existence because an application that is portable must be written to look for (and usually ignore) those entries. Writing code that presumes that they are the first two entries does not always work, as many implementations permit them to be other than the first two entries, with a ``normal'' entry preceding them. There is negligible value in providing a way to determine what the implementation does because the code to deal with dot and dot-dot must be written in any case and because such a flag would add to the list of those flags (which has proven in itself to be objectionable) and might be abused.

Since the structure and buffer allocation, if any, for directory operations are defined by the implementation, this volume of POSIX.1-2024 imposes no portability requirements for erroneous program constructs, erroneous data, or the use of unspecified values such as the use or referencing of a *dirp* value or a **dirent** structure value after a directory stream has been closed or after a *fork*( ) or one of the *exec* function calls.

FUTURE DIRECTIONS

None.

SEE ALSO

closedir( ), *dirfd*( ), *fstatat*( ), *open*( ), *posix_getdents*( ), *readdir*( ), *rewinddir*( ), *symlink*( )

XBD **<dirent.h>**, **<fcntl.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 2.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

The *fdopendir*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

An additional example is added.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0122 [422] and XSH/TC1-2008/0123 [324] are applied.

Issue 8

Austin Group Defect 368 is applied, adding a requirement for FD_CLOEXEC to be set by *opendir*( ) if it associates a file descriptor with the returned stream.

Austin Group Defect 411 is applied, clarifying that FD_CLOEXEC is not cleared by *fdopendir*( ) if it was previously set.

Austin Group Defect 697 is applied, adding *posix_getdents*( ) to the SEE ALSO section.

Austin Group Defect 1360 is applied, clarifying that type **DIR** always has the ability to store a file descriptor; what is optional is whether one is opened by *opendir*( ).

NAME

feclearexcept — clear floating-point exception

SYNOPSIS

```
#include <fenv.h>

int feclearexcept(int excepts);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *feclearexcept*( ) function shall attempt to clear the supported floating-point exceptions represented by *excepts*.

RETURN VALUE

If the argument is zero or if all the specified exceptions were successfully cleared, *feclearexcept*( ) shall return zero. Otherwise, it shall return a non-zero value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fegetexceptflag( ), *feraiseexcept*( ), *fetestexcept*( )

XBD **<fenv.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

NAME

fegetenv, fesetenv — get and set current floating-point environment

SYNOPSIS

```
#include <fenv.h>

int fegetenv(fenv_t *envp);
int fesetenv(const fenv_t *envp);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fegetenv*( ) function shall attempt to store the current floating-point environment in the object pointed to by *envp*.

The *fesetenv*( ) function shall attempt to establish the floating-point environment represented by the object pointed to by *envp*. The argument *envp* shall point to an object set by a call to *fegetenv*( ) or *feholdexcept*( ), or equal a floating-point environment macro. The *fesetenv*( ) function does not raise floating-point exceptions, but only installs the state of the floating-point status flags represented through its argument.

RETURN VALUE

If the representation was successfully stored, *fegetenv*( ) shall return zero. Otherwise, it shall return a non-zero value. If the environment was successfully established, *fesetenv*( ) shall return zero. Otherwise, it shall return a non-zero value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feholdexcept( ), *feupdateenv*( )

XBD **<fenv.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

NAME

fegetexceptflag, fesetexceptflag — get and set floating-point status flags

SYNOPSIS

```
#include <fenv.h>

int fegetexceptflag(fexcept_t *flagp, int excepts);
int fesetexceptflag(const fexcept_t *flagp, int excepts);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fegetexceptflag*( ) function shall attempt to store an implementation-defined representation of the states of the floating-point status flags indicated by the argument *excepts* in the object pointed to by the argument *flagp*.

The *fesetexceptflag*( ) function shall attempt to set the floating-point status flags indicated by the argument *excepts* to the states stored in the object pointed to by *flagp*. The value pointed to by *flagp* shall have been set by a previous call to *fegetexceptflag*( ) whose second argument represented at least those floating-point exceptions represented by the argument *excepts*. This function does not raise floating-point exceptions, but only sets the state of the flags.

RETURN VALUE

If the representation was successfully stored, *fegetexceptflag*( ) shall return zero. Otherwise, it shall return a non-zero value. If the *excepts* argument is zero or if all the specified exceptions were successfully set, *fesetexceptflag*( ) shall return zero. Otherwise, it shall return a non-zero value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *feraiseexcept*( ), *fetestexcept*( )

XBD **<fenv.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

NAME

fegetround, fesetround — get and set current rounding direction

SYNOPSIS

```
#include <fenv.h>

int fegetround(void);
int fesetround(int round);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fegetround*( ) function shall get the current rounding direction.

The *fesetround*( ) function shall establish the rounding direction represented by its argument *round*. If the argument is not equal to the value of a rounding direction macro, the rounding direction is not changed.

RETURN VALUE

The *fegetround*( ) function shall return the value of the rounding direction macro representing the current rounding direction or a negative value if there is no such rounding direction macro or the current rounding direction is not determinable.

The *fesetround*( ) function shall return a zero value if and only if the requested rounding direction was established.

ERRORS

No errors are defined.

EXAMPLES

The following example saves, sets, and restores the rounding direction, reporting an error and aborting if setting the rounding direction fails:

```
#include <fenv.h>
#include <assert.h>
void f(int round_dir)
{
    #pragma STDC FENV_ACCESS ON
    int save_round;
    int setround_ok;
    save_round = fegetround();
    setround_ok = fesetround(round_dir);
    assert(setround_ok == 0);
    /* ... */
    fesetround(save_round);
    /* ... */
}
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD **<fenv.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

NAME

feholdexcept — save current floating-point environment

SYNOPSIS

```
#include <fenv.h>

int feholdexcept(fenv_t *envp);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *feholdexcept*( ) function shall save the current floating-point environment in the object pointed to by *envp*, clear the floating-point status flags, and then install a non-stop (continue on floating-point exceptions) mode, if available, for all floating-point exceptions.

RETURN VALUE

The *feholdexcept*( ) function shall return zero if and only if non-stop floating-point exception handling was successfully installed.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *feholdexcept*( ) function should be effective on typical IEC 60559:1989 standard implementations which have the default non-stop mode and at least one other mode for trap handling or aborting. If the implementation provides only the non-stop mode, then installing the non-stop mode is trivial.

FUTURE DIRECTIONS

None.

SEE ALSO

fegetenv( ), *feupdateenv*( )

XBD **<fenv.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

NAME

feof — test end-of-file indicator on a stream

SYNOPSIS

```
#include <stdio.h>

int feof(FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *feof*( ) function shall test the end-of-file indicator for the stream pointed to by *stream*.

CX The *feof*( ) function shall not change the setting of *errno* if *stream* is valid.

RETURN VALUE

The *feof*( ) function shall return non-zero if and only if the end-of-file indicator is set for *stream*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

clearerr( ), *ferror*( ), *fopen*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0124 [401] is applied.

NAME

feraiseexcept — raise floating-point exception

SYNOPSIS

```
#include <fenv.h>

int feraiseexcept(int excepts);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *feraiseexcept*( ) function shall attempt to raise the supported floating-point exceptions represented by the *excepts* argument. The order in which these floating-point exceptions are raised is unspecified, MX except that if the *excepts* argument represents IEC 60559 valid coincident floating-point exceptions for atomic operations (namely overflow and inexact, or underflow and inexact), then overflow or underflow shall be raised before inexact. Whether the *feraiseexcept*( ) function additionally raises the inexact floating-point exception whenever it raises the overflow or underflow floating-point exception is implementation-defined.

RETURN VALUE

If the argument is zero or if all the specified exceptions were successfully raised, *feraiseexcept*( ) shall return zero. Otherwise, it shall return a non-zero value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The effect is intended to be similar to that of floating-point exceptions raised by arithmetic operations. Hence, enabled traps for floating-point exceptions raised by this function are taken.

RATIONALE

Raising overflow or underflow is allowed to also raise inexact because on some architectures the only practical way to raise an exception is to execute an instruction that has the exception as a side-effect. The function is not restricted to accept only valid coincident expressions for atomic operations, so the function can be used to raise exceptions accrued over several operations.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fegetexceptflag*( ), *fetestexcept*( )

XBD **<fenv.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0111 [543] is applied.

NAME

ferror — test error indicator on a stream

SYNOPSIS

```
#include <stdio.h>

int ferror(FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *ferror*( ) function shall test the error indicator for the stream pointed to by *stream*.

CX The *ferror*( ) function shall not change the setting of *errno* if *stream* is valid.

RETURN VALUE

The *ferror*( ) function shall return non-zero if and only if the error indicator is set for *stream*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

clearerr( ), *feof*( ), *fopen*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0125 [401] is applied.

31845 **NAME**

31846 fesetenv — set current floating-point environment

31847 **SYNOPSIS**

31848 `#include <fenv.h>`

31849 `int fesetenv(const fenv_t *envp);`

31850 **DESCRIPTION**

31851 Refer to *fegetenv( )*.

31852 **NAME**

31853 fesetexceptflag — set floating-point status flags

31854 **SYNOPSIS**

31855 ```
#include <fenv.h>
```

31856 ```
int fesetexceptflag(const fexcept_t *flagp, int excepts);
```

31857 **DESCRIPTION**

31858 Refer to *fegetexceptflag( )*.

31859 **NAME**

31860 fesetround — set current rounding direction

31861 **SYNOPSIS**

31862 `#include <fenv.h>`

31863 `int fesetround(int` *round*`);`

31864 **DESCRIPTION**

31865 Refer to *fegetround( )*.

NAME

fetestexcept — test floating-point exception flags

SYNOPSIS

```
#include <fenv.h>

int fetestexcept(int excepts);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fetestexcept*( ) function shall determine which of a specified subset of the floating-point exception flags are currently set. The *excepts* argument specifies the floating-point status flags to be queried.

RETURN VALUE

The *fetestexcept*( ) function shall return the value of the bitwise-inclusive OR of the floating-point exception macros corresponding to the currently set floating-point exceptions included in *excepts*.

ERRORS

No errors are defined.

EXAMPLES

The following example calls function *f*( ) if an invalid exception is set, and then function *g*( ) if an overflow exception is set:

```
#include <fenv.h>
/* ... */
{
    #pragma STDC FENV_ACCESS ON
    int set_excepts;
    feclearexcept(FE_INVALID | FE_OVERFLOW);
    // maybe raise exceptions
    set_excepts = fetestexcept(FE_INVALID | FE_OVERFLOW);
    if (set_excepts & FE_INVALID) f();
    if (set_excepts & FE_OVERFLOW) g();
    /* ... */
}
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fegetexceptflag*( ), *feraiseexcept*( )

XBD **<fenv.h>**

31908 **CHANGE HISTORY**

31909 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

NAME

feupdateenv — update floating-point environment

SYNOPSIS

```
#include <fenv.h>

int feupdateenv(const fenv_t *envp);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *feupdateenv*( ) function shall attempt to save the currently raised floating-point exceptions in its automatic storage, attempt to install the floating-point environment represented by the object pointed to by *envp*, and then attempt to raise the saved floating-point exceptions. The argument *envp* shall point to an object set by a call to *feholdexcept*( ) or *fegetenv*( ), or equal a floating-point environment macro.

RETURN VALUE

The *feupdateenv*( ) function shall return a zero value if and only if all the required actions were successfully carried out.

ERRORS

No errors are defined.

EXAMPLES

The following example shows sample code to hide spurious underflow floating-point exceptions:

```
#include <fenv.h>
double f(double x)
{
    #pragma STDC FENV_ACCESS ON
    double result;
    fenv_t save_env;
    feholdexcept(&save_env);
    // compute result
    if (/* test spurious underflow */)
    feclearexcept(FE_UNDERFLOW);
    feupdateenv(&save_env);
    return result;
}
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fegetenv( ), *feholdexcept*( )

XBD **<fenv.h>**

31954 **CHANGE HISTORY**

31955 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

31956 ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

NAME

fexecve — execute a file

SYNOPSIS

```
#include <unistd.h>

int fexecve(int fd, char *const argv[], char *const envp[]);
```

DESCRIPTION

Refer to *exec*.

NAME

fflush — flush a stream

SYNOPSIS

```
#include <stdio.h>

int fflush(FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

If *stream* points to an output stream or an update stream in which the most recent operation was not input, *fflush*( ) shall cause any unwritten data for that stream to be written to the file, CX and the last data modification and last file status change timestamps of the underlying file shall be marked for update.

For a stream open for reading with an underlying file description, if the file is not already at EOF, and the file is one capable of seeking, the file offset of the underlying open file description shall be set to the file position of the stream, and any characters pushed back onto the stream by *ungetc*( ) or *ungetwc*( ) that have not subsequently been read from the stream shall be discarded (without further changing the file offset).

If *stream* is a null pointer, *fflush*( ) shall perform this flushing action on all streams for which the behavior is defined above.

RETURN VALUE

Upon successful completion, *fflush*( ) shall return 0; otherwise, it shall set the error indicator for the stream, return EOF, CX and set *errno* to indicate the error.

ERRORS

The *fflush*( ) function shall fail if:

CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the thread would be delayed in the write operation.

CX [EBADF] The file descriptor underlying *stream* is not valid.

CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

CX [EFBIG] An attempt was made to write a file that exceeds the file size limit of the process.
XSI A SIGXFSZ signal shall also be generated for the thread.

CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.

CX [EINTR] The *fflush*( ) function was interrupted by a signal.

CX [EIO] The process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the calling thread is not blocking SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.

CX [ENOMEM] The underlying stream was created by *open_memstream*( ) or *open_wmemstream*( ) and insufficient memory is available.

CX [ENOSPC] There was no free space remaining on the device containing the file or in the buffer used by the *fmemopen*( ) function.

CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal shall also be sent to the thread.

The *fflush*( ) function may fail if:

CX [ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.

EXAMPLES

Sending Prompts to Standard Output

The following example uses *printf*( ) calls to print a series of prompts for information the user must enter from standard input. The *fflush*( ) calls force the output to standard output. The *fflush*( ) function is used because standard output is usually buffered and the prompt may not immediately be printed on the output or terminal. The *getline*( ) function calls read strings from standard input and place the results in variables, for use later in the program.

```
char *user;
char *oldpasswd;
char *newpasswd;
ssize_t llen;
size_t blen;
struct termios term;
tcflag_t saveflag;

printf("User name: ");
fflush(stdout);
blen = 0;
llen = getline(&user, &blen, stdin);
user[llen-1] = 0;
tcgetattr(fileno(stdin), &term);
saveflag = term.c_lflag;
term.c_lflag &= ~ECHO;
tcsetattr(fileno(stdin), TCSANOW, &term);
printf("Old password: ");
fflush(stdout);
blen = 0;
llen = getline(&oldpasswd, &blen, stdin);
oldpasswd[llen-1] = 0;

printf("\nNew password: ");
fflush(stdout);
blen = 0;
llen = getline(&newpasswd, &blen, stdin);
newpasswd[llen-1] = 0;
term.c_lflag = saveflag;
tcsetattr(fileno(stdin), TCSANOW, &term);
free(user);
free(oldpasswd);
free(newpasswd);
```


APPLICATION USAGE

None.

RATIONALE

Data buffered by the system may make determining the validity of the position of the current file descriptor impractical. Thus, enforcing the repositioning of the file descriptor after *fflush*( ) on streams open for *read*( ) is not mandated by POSIX.1-2024.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fmemopen*( ), *getrlimit*( ), *open_memstream*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EFBIG] error is added as part of the large file support extensions.
- The [ENXIO] optional error condition is added.

The RETURN VALUE section is updated to note that the error indicator shall be set for the stream. This is for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/31 is applied, updating the [EAGAIN] error in the ERRORS section from ``the process would be delayed'' to ``the thread would be delayed''.

Issue 7

Austin Group Interpretation 1003.1-2001 #002 is applied, clarifying the interaction of file descriptors and streams.

The [ENOSPC] error condition is updated and the [ENOMEM] error is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The EXAMPLES section is revised.

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0126 [87], XSH/TC1-2008/0127 [79], and XSH/TC1-2008/0128 [14] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0112 [816] and XSH/TC2-2008/0113 [626] are applied.

Issue 8

Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error relating to the file size limit for the process.

32092 **NAME**

32093 ffs, ffsl, ffsll — find first set bit

32094 **SYNOPSIS**

32095 XSI
```
#include <strings.h>

int ffs(int i);
int ffsl(long i);
int ffsll(long long i);
```

32099 **DESCRIPTION**

32100 The *ffs*( ), *ffsl*( ), and *ffsll*( ) functions shall find the first bit set (beginning with the least significant
32101 bit) in *i*, and return the index of that bit. Bits are numbered starting at one (the least significant
32102 bit).

32103 **RETURN VALUE**

32104 The *ffs*( ), *ffsl*( ), and *ffsll*( ) functions shall return the index of the first bit set. If *i* is 0, then these
32105 functions shall return 0.

32106 **ERRORS**

32107 No errors are defined.

32108 **EXAMPLES**

32109 None.

32110 **APPLICATION USAGE**

32111 None.

32112 **RATIONALE**

32113 None.

32114 **FUTURE DIRECTIONS**

32115 None.

32116 **SEE ALSO**

32117 XBD **<strings.h>**

32118 **CHANGE HISTORY**

32119 First released in Issue 4, Version 2.

32120 **Issue 5**

32121 Moved from X/OPEN UNIX extension to BASE.

32122 **Issue 8**

32123 Austin Group Defect 617 is applied, adding *ffsl*( ) and *ffsll*( ).

NAME

fgetc — get a byte from a stream

SYNOPSIS

```
#include <stdio.h>

int fgetc(FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

If the end-of-file indicator for the input stream pointed to by *stream* is not set and a next byte is present, the *fgetc*( ) function shall obtain the next byte as an **unsigned char** converted to an **int**, from the input stream pointed to by *stream*, and advance the associated file position indicator for the stream (if defined). Since *fgetc*( ) operates on bytes, reading a character consisting of multiple bytes (or ``a multi-byte character'') may require multiple calls to *fgetc*( ).

CX The *fgetc*( ) function may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetc*( ), *fgets*( ), *fread*( ), *fscanf*( ), *getc*( ), *getchar*( ), *getdelim*( ), *getline*( ), or *scanf*( ) using *stream* that returns data not supplied by a prior call to *ungetc*( ).

RETURN VALUE

Upon successful completion, *fgetc*( ) shall return the next byte from the input stream pointed to by *stream*. If the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and *fgetc*( ) shall return EOF. If an error occurs, the CX error indicator for the stream shall be set, *fgetc*( ) shall return EOF, and shall set *errno* to indicate the error.

ERRORS

The *fgetc*( ) function shall fail if data needs to be read and:

CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the thread would be delayed in the *fgetc*( ) operation.

CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for reading.

CX [EINTR] The read operation was terminated due to the receipt of a signal, and no data was transferred.

CX [EIO] A physical I/O error has occurred, or the process is in a background process group attempting to read from its controlling terminal, and either the calling thread is blocking SIGTTIN or the process is ignoring SIGTTIN or the process group of the process is orphaned. This error may also be generated for implementation-defined reasons.

CX [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the corresponding stream.

The *fgetc*( ) function may fail if:

CX [ENOMEM] Insufficient storage space is available.

CX [ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.

EXAMPLES

None.

APPLICATION USAGE

If the integer value returned by *fgetc*( ) is stored into a variable of type **char** and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a variable of type **char** on widening to integer is implementation-defined.

The *ferror*( ) or *feof*( ) functions must be used to distinguish between an error condition and an end-of-file condition.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *feof*( ), *ferror*( ), *fgets*( ), *fread*( ), *fscanf*( ), *getchar*( ), *getc*( ), *ungetc*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- The [ENOMEM] and [ENXIO] optional error conditions are added.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The DESCRIPTION is updated to clarify the behavior when the end-of-file indicator for the input stream is not set.
- The RETURN VALUE section is updated to note that the error indicator shall be set for the stream.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/32 is applied, updating the [EAGAIN] error in the ERRORS section from ``the process would be delayed'' to ``the thread would be delayed''.

Issue 7

Austin Group Interpretation 1003.1-2001 #051 is applied, updating the list of functions that mark the last data access timestamp for update.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0129 [79] and XSH/TC1-2008/0130 [14] are applied.

Issue 8

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1624 is applied, changing the RETURN VALUE section.

NAME

fgetpos — get current file position information

SYNOPSIS

```
#include <stdio.h>

int fgetpos(FILE *restrict stream, fpos_t *restrict pos);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fgetpos*( ) function shall store the current values of the parse state (if any) and file position indicator for the stream pointed to by *stream* in the object pointed to by *pos*. The value stored contains unspecified information usable by *fsetpos*( ) for repositioning the stream to its position at the time of the call to *fgetpos*( ).

The *fgetpos*( ) function shall not change the setting of *errno* if successful.

RETURN VALUE

Upon successful completion, *fgetpos*( ) shall return 0; otherwise, it shall return a non-zero value and set *errno* to indicate the error.

ERRORS

The *fgetpos*( ) function shall fail if:

CX [EBADF] The file descriptor underlying *stream* is not valid.

CX [EOVERFLOW] The current value of the file position cannot be represented correctly in an object of type **fpos_t**.

CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fopen*( ), *ftell*( ), *rewind*( ), *ungetc*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ISO C standard.

Issue 5

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EBADF] and [ESPIPE] optional error conditions are added.

An additional [ESPIPE] error condition is added for sockets.

The prototype for *fgetpos*( ) is changed for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0131 [105], XSH/TC1-2008/0132 [122], and XSH/TC1-2008/0133 [14] are applied.

NAME

fgets — get a string from a stream

SYNOPSIS

```
#include <stdio.h>

char *fgets(char *restrict s, int n, FILE *restrict stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fgets*( ) function shall read bytes from *stream* into the array pointed to by *s* until *n*–1 bytes are read, or a <newline> is read and transferred to *s*, or an end-of-file condition is encountered. A null byte shall be written immediately after the last byte read into the array. If the end-of-file condition is encountered before any bytes are read, the contents of the array pointed to by *s* shall not be changed.

CX The *fgets*( ) function may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetc*( ), *fgets*( ), *fread*( ), *fscanf*( ), *getc*( ), *getchar*( ), *getdelim*( ), *getline*( ), or *scanf*( ) using *stream* that returns data not supplied by a prior call to *ungetc*( ).

RETURN VALUE

Upon successful completion, *fgets*( ) shall return *s*. If the stream is at end-of-file, the end-of-file indicator for the stream shall be set and *fgets*( ) shall return a null pointer. If an error occurs, the CX error indicator for the stream shall be set, *fgets*( ) shall return a null pointer, and shall set *errno* to indicate the error.

ERRORS

Refer to *fgetc*( ).

EXAMPLES

Reading Input

The following example uses *fgets*( ) to read lines of input. It assumes that the file it is reading is a text file and that lines in this text file are no longer than 16384 (or {LINE_MAX} if it is less than 16384 on the implementation where it is running) bytes long. (Note that the standard utilities have no line length limit if *sysconf*(_SC_LINE_MAX) returns –1 without setting *errno*. This example assumes that *sysconf*(_SC_LINE_MAX) will not fail.)

```
#include <limits.h>
#include <stdio.h>
#include <unistd.h>
#define MYLIMIT 16384

char *line;
int line_max;
if (LINE_MAX >= MYLIMIT) {
    // Use maximum line size of MYLIMIT. If LINE_MAX is
    // bigger than our limit, sysconf() cannot report a
    // smaller limit.
    line_max = MYLIMIT;
} else {
    long limit = sysconf(_SC_LINE_MAX);
    line_max = (limit < 0 || limit > MYLIMIT) ? MYLIMIT : (int)limit;
```



```
    }

    // line_max + 1 leaves room for the null byte added by fgets().
    line = malloc(line_max + 1);
    if (line == NULL) {
        // out of space
        ...
        return error;
    }

    while (fgets(line, line_max + 1, fp) != NULL) {
        // Verify that a full line has been read ...
        // If not, report an error or prepare to treat the
        // next time through the loop as a read of a
        // continuation of the current line.
        ...
        // Process line ...
        ...
    }
    free(line);
    ...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fgetc( )*, *fopen( )*, *fread( )*, *fscanf( )*, *getc( )*, *getchar( )*, *getdelim( )*, *ungetc( )*

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

The prototype for *fgets( )* is changed for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #051 is applied, updating the list of functions that mark the last data access timestamp for update.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0134 [182] and XSH/TC1-2008/0135 [14] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0114 [468] is applied.

Issue 8

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1624 is applied, changing the RETURN VALUE section.

NAME

fgetwc — get a wide-character code from a stream

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

wint_t fgetwc(FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fgetwc*( ) function shall obtain the next character (if present) from the input stream pointed to by *stream*, convert that to the corresponding wide-character code, and advance the associated file position indicator for the stream (if defined).

If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

CX The *fgetwc*( ) function may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetwc*( ), *fgetws*( ), *fwscanf*( ), *getwc*( ), *getwchar*( ), *vfwscanf*( ), *vwscanf*( ), or *wscanf*( ) using *stream* that returns data not supplied by a prior call to *ungetwc*( ).

The *fgetwc*( ) function shall not change the setting of *errno* if successful.

RETURN VALUE

Upon successful completion, the *fgetwc*( ) function shall return the wide-character code of the character read from the input stream pointed to by *stream* converted to a type **wint_t**. If the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and *fgetwc*( ) shall return WEOF. If an error other than an encoding error occurs, the error indicator for the stream shall be set, and *fgetwc*( ) shall return WEOF CX and shall set *errno* to indicate the error. CX If an encoding error occurs, the error indicator for the stream shall be set, and *fgetwc*( ) shall return WEOF and shall set *errno* to indicate the error.

ERRORS

The *fgetwc*( ) function shall fail if data needs to be read and:

CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the thread would be delayed in the *fgetwc*( ) operation.

CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for reading.

[EILSEQ] The data obtained from the input stream does not form a valid character.

CX [EINTR] The read operation was terminated due to the receipt of a signal, and no data was transferred.

CX [EIO] A physical I/O error has occurred, or the process is in a background process group attempting to read from its controlling terminal, and either the calling thread is blocking SIGTTIN or the process is ignoring SIGTTIN or the process group of the process is orphaned. This error may also be generated for implementation-defined reasons.

CX [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the corresponding stream.

The *fgetwc*( ) function may fail if:

CX [ENOMEM] Insufficient storage space is available.

CX [ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.

EXAMPLES

None.

APPLICATION USAGE

The *ferror*( ) or *feof*( ) functions must be used to distinguish between an error condition and an end-of-file condition.

RATIONALE

The requirement to set the error indicator when an encoding error occurs is shaded CX because this is not required by the ISO C standard. However, the next revision of the ISO C standard is expected to add this requirement.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *feof*( ), *ferror*( ), *fopen*( )

XBD **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

The Optional Header (OH) marking is removed from **<stdio.h>**.

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- The [ENOMEM] and [ENXIO] optional error conditions are added.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/33 is applied, updating the [EAGAIN] error in the ERRORS section from ``the process would be delayed'' to ``the thread would be delayed''.

Issue 7

Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0136 [105], XSH/TC1-2008/0137 [79], and XSH/TC1-2008/0138 [14] are applied.

Issue 8

Austin Group Defect 1624 is applied, changing the RETURN VALUE and RATIONALE sections.

NAME

fgetws — get a wide-character string from a stream

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

wchar_t *fgetws(wchar_t *restrict ws, int n,
    FILE *restrict stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fgetws*( ) function shall read characters from the *stream*, convert these to the corresponding wide-character codes, place them in the **wchar_t** array pointed to by *ws*, until *n*–1 characters are read, or a **<**newline**>** is read, converted, and transferred to *ws*, or an end-of-file condition is encountered. The wide-character string, *ws*, shall then be terminated with a null wide-character code.

If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

CX The *fgetws*( ) function may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetwc*( ), *fgetws*( ), *fwscanf*( ), *getwc*( ), *getwchar*( ), *vfwscanf*( ), *vwscanf*( ), or *wscanf*( ) using *stream* that returns data not supplied by a prior call to *ungetwc*( ).

RETURN VALUE

Upon successful completion, *fgetws*( ) shall return *ws*. If the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and *fgetws*( ) shall return a null pointer. If an error occurs, the error indicator for the stream shall be set, and *fgetws*( ) shall return a null pointer CX and shall set *errno* to indicate the error.

ERRORS

Refer to *fgetwc*( ).

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fopen*( ), *fread*( )

XBD **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

The Optional Header (OH) marking is removed from **<stdio.h>**.

Issue 6

Extensions beyond the ISO C standard are marked.

The prototype for *fgetws*( ) is changed for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0139 [14] is applied.

Issue 8

Austin Group Defect 1624 is applied, changing the RETURN VALUE section.

NAME

fileno — map a stream pointer to a file descriptor

SYNOPSIS

CX
```
#include <stdio.h>

int fileno(FILE *stream);
```

DESCRIPTION

The *fileno*( ) function shall return the integer file descriptor associated with the stream pointed to by *stream*.

RETURN VALUE

Upon successful completion, *fileno*( ) shall return the integer value of the file descriptor associated with *stream*. Otherwise, the value –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *fileno*( ) function shall fail if:

[EBADF] The stream is not associated with a file.

The *fileno*( ) function may fail if:

[EBADF] The file descriptor underlying *stream* is not a valid file descriptor.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Without some specification of which file descriptors are associated with these streams, it is impossible for an application to set up the streams for another application it starts with *fork*( ) and *exec*. In particular, it would not be possible to write a portable version of the *sh* command interpreter (although there may be other constraints that would prevent that portability).

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5.1 (on page 522), *dirfd*( ), *fdopen*( ), *fopen*( ), *stdin*

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EBADF] optional error condition is added.

Issue 7

SD5-XBD-ERN-99 is applied, changing the definition of the [EBADF] error.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0115 [589] is applied.

NAME

flockfile, ftrylockfile, funlockfile — stdio locking functions

SYNOPSIS

CX
```
#include <stdio.h>

void flockfile(FILE *file);
int ftrylockfile(FILE *file);
void funlockfile(FILE *file);
```

DESCRIPTION

These functions shall provide for explicit application-level locking of the locks associated with standard I/O streams (see Section 2.5, on page 521). These functions can be used by a thread to delineate a sequence of I/O statements that are executed as a unit.

The *flockfile*( ) function shall acquire for a thread ownership of a (**FILE** *) object.

The *ftrylockfile*( ) function shall acquire for a thread ownership of a (**FILE** *) object if the object is available; *ftrylockfile*( ) is a non-blocking version of *flockfile*( ).

The *funlockfile*( ) function shall relinquish the ownership granted to the thread. The behavior is undefined if a thread other than the current owner calls the *funlockfile*( ) function.

The functions shall behave as if there is a lock count associated with each (**FILE** *) object. This count is implicitly initialized to zero when the (**FILE** *) object is created. The (**FILE** *) object is unlocked when the count is zero. When the count is positive, a single thread owns the (**FILE** *) object. When the *flockfile*( ) function is called, if the count is zero or if the count is positive and the caller owns the (**FILE** *) object, the count shall be incremented. Otherwise, the calling thread shall be suspended, waiting for the count to return to zero. Each call to *funlockfile*( ) shall decrement the count. This allows matching calls to *flockfile*( ) (or successful calls to *ftrylockfile*( )) and *funlockfile*( ) to be nested.

RETURN VALUE

None for *flockfile*( ) and *funlockfile*( ).

The *ftrylockfile*( ) function shall return zero for success and non-zero to indicate that the lock cannot be acquired.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in XBD Section 3.275 (on page 72).

A call to *exit*( ) can block until locked streams are unlocked because a thread having ownership of a (**FILE***) object blocks all function calls that reference that (**FILE***) object (except those with names ending in _unlocked) from other threads, including calls to *exit*( ).

Note: a **FILE** lock is not a file lock (see XBD Section 3.143, on page 51).

RATIONALE

The *flockfile*( ) and *funlockfile*( ) functions provide an orthogonal mutual-exclusion lock for each **FILE**. The *ftrylockfile*( ) function provides a non-blocking attempt to acquire a **FILE** lock, analogous to *pthread_mutex_trylock*( ).

These locks behave as if they are the same as those used internally by *stdio* for thread-safety.

This both provides thread-safety of these functions without requiring a second level of internal locking and allows functions in *stdio* to be implemented in terms of other *stdio* functions.

Application developers and implementors should be aware that there are potential deadlock problems on **FILE** objects. For example, the line-buffered flushing semantics of *stdio* (requested via {_IOLBF}) require that certain input operations sometimes cause the buffered contents of implementation-defined line-buffered output streams to be flushed. If two threads each hold the lock on the other's **FILE**, deadlock ensues. This type of deadlock can be avoided by acquiring **FILE** locks in a consistent order. In particular, the line-buffered output stream deadlock can typically be avoided by acquiring locks on input streams before locks on output streams if a thread would be acquiring both.

In summary, threads sharing *stdio* streams with other threads can use *flockfile*( ) and *funlockfile*( ) to cause sequences of I/O performed by a single thread to be kept bundled. The only case where the use of *flockfile*( ) and *funlockfile*( ) is required is to provide a scope protecting uses of the **_unlocked* functions/macros. This moves the cost/performance tradeoff to the optimal point.

FUTURE DIRECTIONS

None.

SEE ALSO

exit( ), *getc_unlocked*( )

XBD Section 3.275 (on page 72), **<stdio.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

These functions are marked as part of the Thread-Safe Functions option.

Issue 7

The *flockfile*( ), *ftrylockfile*( ), and *funlockfile*( ) functions are moved from the Thread-Safe Functions option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0140 [118] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0116 [611] is applied.

Issue 8

Austin Group Defect 1118 is applied, clarifying that a **FILE** lock is not a file lock.

Austin Group Defect 1302 is applied, replacing parts of the text with a reference to Section 2.5 (on page 521).

NAME

floor, floorf, floorl — floor function

SYNOPSIS

```
#include <math.h>

double floor(double x);
float floorf(float x);
long double floorl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the largest integral value not greater than x.

MX These functions may raise the inexact floating-point exception for finite non-integer arguments.

RETURN VALUE

MX The returned value shall be independent of the current rounding direction mode and shall have the same sign as x.

Upon successful completion, these functions shall return the largest integral value not greater than x, expressed as a **double**, **float**, or **long double**, as appropriate for the return type of the function.

MX If x is NaN, a NaN shall be returned.

If x is ±0 or ±Inf, x shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The integral value returned by these functions might not be expressible as an **intmax_t**. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ceil( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( )

Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *floorf*( ) and *floorl*( ) functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0141 [346] is applied.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

NAME

fma, fmaf, fmal — floating-point multiply-add

SYNOPSIS

```
#include <math.h>

double fma(double x, double y, double z);
float fmaf(float x, float y, float z);
long double fmal(long double x, long double y, long double z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute $(x * y) + z$, rounded as one ternary operation: they shall compute the value (as if) to infinite precision and round once to the result format, according to the rounding mode characterized by the value of FLT_ROUNDS.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return $(x * y) + z$, rounded as one ternary operation.

MX If the result overflows or underflows, a range error may occur. On systems that support the IEC 60559 Floating-Point option, if the result overflows a range error shall occur.

If x or y are NaN, a NaN shall be returned.

If x multiplied by y is an exact infinity and z is also an infinity but with the opposite sign, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If one of x and y is infinite, the other is zero, and z is not a NaN, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If one of x and y is infinite, the other is zero, and z is a NaN, a NaN shall be returned and a domain error may occur.

If $x*y$ is not 0*Inf nor Inf*0 and z is a NaN, a NaN shall be returned.

ERRORS

These functions shall fail if:

MX **Domain Error** The value of $x*y+z$ is invalid, or the value $x*y$ is invalid and z is not a NaN.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

MX **Range Error** The result overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

MX

Domain Error	The value $x*y$ is invalid and z is a NaN. If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.
Range Error	The result underflows. If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.
Range Error	The result overflows. If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

In many cases, clever use of floating (*fused*) multiply-add leads to much improved code; but its unexpected use by the compiler can undermine carefully written code. The FP_CONTRACT macro can be used to disallow use of floating multiply-add; and the *fma*( ) function guarantees its use where desired. Many current machines provide hardware floating multiply-add instructions; software implementation can be used for others.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #57 (SD5-XSH-ERN-69) is applied, adding a ``may fail'' range error for non-MX systems.

NAME

fmax, fmaxf, fmaxl — determine maximum numeric value of two floating-point numbers

SYNOPSIS

```
#include <math.h>

double fmax(double x, double y);
float fmaxf(float x, float y);
long double fmaxl(long double x, long double y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

MX These functions shall determine the maximum numeric value of their arguments. NaN arguments shall be treated as missing data: if one argument is a NaN and the other numeric, then these functions shall choose the numeric value.

RETURN VALUE

Upon successful completion, these functions shall return the maximum numeric value of their arguments.

MX The returned value shall be exact and shall be independent of the current rounding direction mode.

If just one argument is a NaN, the other argument shall be returned.

If *x* and *y* are NaN, a NaN shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fdim( ), *fmin*( )

XBD **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #007 is applied.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

32768 **NAME**

32769 fmemopen — open a memory buffer stream

32770 **SYNOPSIS**

32771 CX
```
#include <stdio.h>

FILE *fmemopen(void *restrict buf, size_t max_size,
    const char *restrict mode);
```

32774 **DESCRIPTION**

32775 The *fmemopen*( ) function shall associate the buffer given by the *buf* and *max_size* arguments with
32776 a stream. The *buf* argument shall be either a null pointer or point to a buffer that is at least
32777 *max_size* bytes long.

32778 The *mode* argument points to a string. If the string is one of the following, the stream shall be
32779 opened in the indicated mode. Otherwise, the behavior is undefined.

32780 *r* or *rb* Open the stream for reading.

32781 *w* or *wb* Open the stream for writing.

32782 *a* or *ab* Append; open the stream for writing.

32783 *r+* or *rb+* or *r+b* Open the stream for update (reading and writing).

32784 *w+* or *wb+* or *w+b* Open the stream for update (reading and writing).

32785 *a+* or *ab+* or *a+b* Append; open the stream for update (reading and writing).

32786 If the *mode* argument begins with `'w'` and *max_size* is not zero, the buffer contents shall be
32787 truncated by writing a null byte at the beginning. If the *mode* argument includes `'b'`, the results
32788 are implementation-defined.

32789 If a null pointer is specified as the *buf* argument, *fmemopen*( ) shall allocate *max_size* bytes of
32790 memory as if by a call to *malloc*( ). This buffer shall be automatically freed when the stream is
32791 closed. Because this feature is only useful when the stream is opened for updating (because
32792 there is no way to get a pointer to the buffer) the *fmemopen*( ) call may fail if the *mode* argument
32793 does not include a `'+'`.

32794 When a stream is opened for reading only and *buf* is not a null pointer, the buffer pointed to by
32795 *buf* shall not be modified by any operation performed on the stream.

32796 The stream shall maintain a current position in the buffer. This position shall be initially set to
32797 either the beginning of the buffer (for *r* and *w* modes) or to the first null byte in the buffer (for *a*
32798 modes). If no null byte is found in append mode, the initial position shall be set to one byte after
32799 the end of the buffer.

32800 If *buf* is a null pointer, the initial position shall always be set to the beginning of the buffer.

32801 The stream shall also maintain the end position of the current buffer contents; use of *fseek*( ) or
32802 *fseeko*( ) on the stream with SEEK_END shall seek relative to this end position. If *mode* starts
32803 with `'r'`, the end position shall be set to the value given by the *max_size* argument and shall not
32804 change. Otherwise, the stream is writable and the end position shall be variable; for modes *w*
32805 and *w+* the initial end position shall be zero and for modes *a* and *a+* the initial end position shall
32806 be:

32807 • Zero, if *buf* is a null pointer

- The position of the first null byte in the buffer, if one is found
- The value of the *max_size* argument, if *buf* is not a null pointer and no null byte is found

A read operation on the stream shall not advance the current buffer position beyond the current buffer end position. Reaching the buffer end position in a read operation shall count as ``end-of-file''. Null bytes in the buffer shall have no special meaning for reads. The read operation shall start at the current buffer position of the stream.

A write operation shall start either at the current position of the stream (if *mode* has not specified `'a'` as the first character) or at the current end position of the stream (if *mode* had `'a'` as the first character). If the current position at the end of the write is larger than the current buffer end position, the current buffer end position shall be set to the current position. A write operation on the stream shall not advance the current buffer end position beyond the size given in the *max_size* argument.

When a stream open for update (the *mode* argument includes `'+'`) or for writing only is successfully written and the write advances the current buffer end position, a null byte shall be written at the new buffer end position if it fits.

An attempt to seek a memory buffer stream to a negative position or to a position larger than the buffer size given in the *max_size* argument shall fail.

RETURN VALUE

Upon successful completion, *fmemopen*( ) shall return a pointer to the object controlling the stream. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

ERRORS

The *fmemopen*( ) function shall fail if:

[EMFILE] {STREAM_MAX} streams are currently open in the calling process.

The *fmemopen*( ) function may fail if:

[EINVAL] The value of the *mode* argument is not valid.

[EINVAL] The *buf* argument is a null pointer and the *mode* argument does not include a `'+'` character.

[EINVAL] The *max_size* argument specifies a buffer size of zero and the implementation does not support this.

[ENOMEM] The *buf* argument is a null pointer and the allocation of a buffer of length *max_size* has failed.

[EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

EXAMPLES

```
#include <stdio.h>
#include <string.h>

static char buffer[] = "foobar";

int
main (void)
{
    int ch;
    FILE *stream;

    stream = fmemopen(buffer, strlen (buffer), "r");
```



```
        if (stream == NULL)
            /* handle error */;
        while ((ch = fgetc(stream)) != EOF)
            printf("Got %c\n", ch);
        fclose(stream);
        return (0);
    }
```

This program produces the following output:

```
Got f
Got o
Got o
Got b
Got a
Got r
```

APPLICATION USAGE

Implementations differ as regards how a `'b'` in the *mode* argument affects the behavior. For some the `'b'` has no effect, as is required for *fopen*( ); others distinguish between text and binary modes.

Note that *buf* will not be null terminated if *max_size* bytes are written to the memory stream. Applications wanting to guarantee that the buffer will be null terminated need to call *fmemopen*( ) with *max_size* set to one byte smaller than the actual size of *buf* and set *buf*[*max_size*] to a null byte.

This standard intentionally leaves the behavior of `'e'` and `'x'` in the *mode* argument undefined; implementations might silently ignore them so that *fmemopen*( ) may accept the same mode strings as *fopen*( ), or may reject them as invalid.

RATIONALE

This interface has been introduced to eliminate many of the errors encountered in the construction of strings, notably overflowing of strings. This interface prevents overflow.

FUTURE DIRECTIONS

A future version of this standard may require support of zero-length buffer streams explicitly.

SEE ALSO

fdopen( ), *fopen*( ), *freopen*( ), *fseek*( ), *malloc*( ), *open_memstream*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 7.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0142 [461], XSH/TC1-2008/0143 [396], XSH/TC1-2008/0144 [396], XSH/TC1-2008/0145 [461], XSH/TC1-2008/0146 [461], XSH/TC1-2008/0147 [461], XSH/TC1-2008/0148 [461], XSH/TC1-2008/0149 [461], and XSH/TC1-2008/0150 [396] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0117 [587], XSH/TC2-2008/0118 [586,818], and XSH/TC2-2008/0119 [818] are applied.

Issue 8

Austin Group Defects 456 and 657 are applied, making the behavior implementation-defined when the *mode* argument includes `'b'`.

Austin Group Defect 1144 is applied, adding a requirement that operations on streams opened for reading only do not modify the buffer pointed to by *buf*.

NAME

fmin, fminf, fminl — determine minimum numeric value of two floating-point numbers

SYNOPSIS

```
#include <math.h>

double fmin(double x, double y);
float fminf(float x, float y);
long double fminl(long double x, long double y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

MX These functions shall determine the minimum numeric value of their arguments. NaN arguments shall be treated as missing data: if one argument is a NaN and the other numeric, then these functions shall choose the numeric value.

RETURN VALUE

Upon successful completion, these functions shall return the minimum numeric value of their arguments.

MX The returned value shall be exact and shall be independent of the current rounding direction mode.

If just one argument is a NaN, the other argument shall be returned.

If *x* and *y* are NaN, a NaN shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fdim( ), *fmax*( )

XBD **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #008 is applied.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

NAME

fmod, fmodf, fmodl — floating-point remainder value function

SYNOPSIS

```
#include <math.h>

double fmod(double x, double y);
float fmodf(float x, float y);
long double fmodl(long double x, long double y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall return the floating-point remainder of the division of *x* by *y*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

These functions shall return the value $x-i*y$, for some integer i such that, if y is non-zero, the result has the same sign as x and magnitude less than the magnitude of y.

MXX If the correct value would cause underflow, and is not representable, a range error may occur, and *fmod*( ), *modf*( ), and *fmodl*( ) shall return 0.0, or (if the IEC 60559 Floating-Point option is not supported) an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

MX If *x* or *y* is NaN, a NaN shall be returned, and none of the conditions below shall be considered.

If *y* is zero, a domain error shall occur, and a NaN shall be returned.

If *x* is infinite, a domain error shall occur, and a NaN shall be returned.

If *x* is ±0 and *y* is not zero, ±0 shall be returned.

If *x* is not infinite and *y* is ±Inf, *x* shall be returned.

When subnormal results are supported, the returned value shall be exact and shall be independent of the current rounding direction mode.

ERRORS

These functions shall fail if:

MX Domain Error — The *x* argument is infinite or *y* is zero.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Range Error — The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( )

Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The behavior for when the *y* argument is zero is now defined.

The *fmodf*( ) and *fmodl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0151 [68], XSH/TC1-2008/0152 [320], and XSH/TC1-2008/0153 [68] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0120 [605] is applied.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018 standard.

NAME

fmtmsg — display a message in the specified format on standard error and/or a system console

SYNOPSIS

XSI

```
#include <fmtmsg.h>

int fmtmsg(long classification, const char *label, int severity,
    const char *text, const char *action, const char *tag);
```

DESCRIPTION

The *fmtmsg*( ) function shall display messages in a specified format instead of the traditional *printf*( ) function.

Based on a message's classification component, *fmtmsg*( ) shall write a formatted message either to standard error, to the console, or to both.

A formatted message consists of up to five components as defined below. The component *classification* is not part of a message displayed to the user, but defines the source of the message and directs the display of the formatted message.

classification Contains the sum of identifying values constructed from the constants defined below. Any one identifier from a subclass may be used in combination with a single identifier from a different subclass. Two or more identifiers from the same subclass should not be used together, with the exception of identifiers from the display subclass. (Both display subclass identifiers may be used so that messages can be displayed to both standard error and the system console.)

Major Classifications

Identifies the source of the condition. Identifiers are: MM_HARD (hardware), MM_SOFT (software), and MM_FIRM (firmware).

Message Source Subclassifications

Identifies the type of software in which the problem is detected. Identifiers are: MM_APPL (application), MM_UTIL (utility), and MM_OPSYS (operating system).

Display Subclassifications

Indicates where the message is to be displayed. Identifiers are: MM_PRINT to display the message on the standard error stream, MM_CONSOLE to display the message on the system console. One or both identifiers may be used.

Status Subclassifications

Indicates whether the application can recover from the condition. Identifiers are: MM_RECOVER (recoverable) and MM_NRECOV (non-recoverable).

An additional identifier, MM_NULLMC, indicates that no classification component is supplied for the message.

label Identifies the source of the message. The format is two fields separated by a <colon>. The first field is up to 10 bytes, the second is up to 14 bytes.

severity Indicates the seriousness of the condition. Identifiers for the levels of *severity* are:

33055 MM_HALT Indicates that the application has encountered a severe fault
33056 and is halting. Produces the string "HALT".

33057 MM_ERROR Indicates that the application has detected a fault. Produces
33058 the string "ERROR".

33059 MM_WARNING Indicates a condition that is out of the ordinary, that might
33060 be a problem, and should be watched. Produces the string
33061 "WARNING".

33062 MM_INFO Provides information about a condition that is not in error.
33063 Produces the string "INFO".

33064 MM_NOSEV Indicates that no severity level is supplied for the message.

33065 *text* Describes the error condition that produced the message. The character string
33066 is not limited to a specific size. If the character string is empty, then the text
33067 produced is unspecified.

33068 *action* Describes the first step to be taken in the error-recovery process. The *fmtmsg*( )
33069 function precedes the action string with the prefix: "TO FIX:". The *action*
33070 string is not limited to a specific size.

33071 *tag* An identifier that references on-line documentation for the message.
33072 Suggested usage is that *tag* includes the *label* and a unique identifying number.
33073 A sample *tag* is "XSI:cat:146".

33074 The *MSGVERB* environment variable (for message verbosity) shall determine for *fmtmsg*( )
33075 which message components it is to select when writing messages to standard error. The value of
33076 *MSGVERB* shall be a <colon>-separated list of optional keywords. Valid keywords are: *label*,
33077 *severity*, *text*, *action*, and *tag*. If *MSGVERB* contains a keyword for a component and the
33078 component's value is not the component's null value, *fmtmsg*( ) shall include that component in
33079 the message when writing the message to standard error. If *MSGVERB* does not include a
33080 keyword for a message component, that component shall not be included in the display of the
33081 message. The keywords may appear in any order. If *MSGVERB* is not defined, if its value is the
33082 null string, if its value is not of the correct format, or if it contains keywords other than the valid
33083 ones listed above, *fmtmsg*( ) shall select all components.

33084 *MSGVERB* shall determine which components are selected for display to standard error. All
33085 message components shall be included in console messages.

33086 **RETURN VALUE**

33087 The *fmtmsg*( ) function shall return one of the following values:

33088 MM_OK The function succeeded.

33089 MM_NOTOK The function failed completely.

33090 MM_NOMSG The function was unable to generate a message on standard error, but
33091 otherwise succeeded.

33092 MM_NOCON The function was unable to generate a console message, but otherwise
33093 succeeded.

33094 **ERRORS**

33095 None.

EXAMPLES

1. The following example of *fmtmsg*( ):

```
fmtmsg(MM_PRINT, "XSI:cat", MM_ERROR, "illegal option",
"refer to cat in user's reference manual", "XSI:cat:001")
```

produces a complete message in the specified message format:

```
XSI:cat: ERROR: illegal option
TO FIX: refer to cat in user's reference manual XSI:cat:001
```

2. When the environment variable *MSGVERB* is set as follows:

```
MSGVERB=severity:text:action
```

and Example 1 is used, *fmtmsg*( ) produces:

```
ERROR: illegal option
TO FIX: refer to cat in user's reference manual
```

APPLICATION USAGE

One or more message components may be systematically omitted from messages generated by an application by using the null value of the argument for that component.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fprintf( )

XBD **<fmtmsg.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

NAME

fnmatch — match a filename string or a pathname

SYNOPSIS

```
#include <fnmatch.h>

int fnmatch(const char *pattern, const char *string, int flags);
```

DESCRIPTION

The *fnmatch*( ) function shall match patterns as described in XCU Section 2.14.1 (on page 2523) and Section 2.14.2 (on page 2524). It checks the string specified by the *string* argument to see if it matches the pattern specified by the *pattern* argument.

The *flags* argument shall modify the interpretation of *pattern* and *string*. It is the bitwise-inclusive OR of zero or more of the flags defined in **<fnmatch.h>**. If the FNM_PATHNAME flag is set in *flags*, then a <slash> character (`'/'`) in *string* shall be explicitly matched by a <slash> in *pattern*; it shall not be matched by either the <asterisk> or <question-mark> special characters, nor by a bracket expression. If the FNM_PATHNAME flag is not set, the <slash> character shall be treated as an ordinary character.

If FNM_NOESCAPE is not set in *flags*, a <backslash> character can be used as an escape character as described in XCU Section 2.14.1 (on page 2523). If *pattern* ends with an unescaped <backslash>, the behavior is unspecified. If FNM_NOESCAPE is set, a <backslash> character shall be treated as an ordinary character.

If FNM_PERIOD is set in *flags*, then a leading <period> (`'.'`) in *string* shall match a <period> in *pattern*; as described by rule 2 in XCU Section 2.14.3 (on page 2525) where the location of ``leading'' is indicated by the value of FNM_PATHNAME:

- If FNM_PATHNAME is set, a <period> is ``leading'' if it is the first character in *string* or if it immediately follows a <slash>.
- If FNM_PATHNAME is not set, a <period> is ``leading'' only if it is the first character of *string*.

If FNM_PERIOD is not set, then no special restrictions are placed on matching a period.

If FNM_CASEFOLD or FNM_IGNORECASE is set, *string* and *pattern* shall be compared in a case-insensitive manner. See XBD Section 4.1 (on page 95).

RETURN VALUE

If *string* matches the pattern specified by *pattern*, then *fnmatch*( ) shall return 0. If there is no match, *fnmatch*( ) shall return FNM_NOMATCH, which is defined in **<fnmatch.h>**. If an error occurs, *fnmatch*( ) shall return another non-zero value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *fnmatch*( ) function has two major uses. It could be used by an application or utility that needs to read a directory and apply a pattern against each entry. The *find* utility is an example of this. It can also be used by the *pax* utility to process its *pattern* operands, or by applications that need to match strings in a similar manner.

The name *fnmatch*( ) is intended to imply *filename* match, rather than *pathname* match. The default action of this function is to match filename strings, rather than pathnames, since it gives no special significance to the <slash> character. With the FNM_PATHNAME flag, *fnmatch*( ) does

match pathnames, but without tilde expansion, parameter expansion, or special treatment for a <period> at the beginning of a filename.

RATIONALE

This function replaced the REG_FILENAME flag of *regcomp*( ) in early proposals of this volume of POSIX.1-2024. It provides virtually the same functionality as the *regcomp*( ) and *regexec*( ) functions using the REG_FILENAME and REG_FSLASH flags (the REG_FSLASH flag was proposed for *regcomp*( ), and would have had the opposite effect from FNM_PATHNAME), but with a simpler function and less system overhead.

FUTURE DIRECTIONS

None.

SEE ALSO

glob( ), *wordexp*( )

XBD **<fnmatch.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 5

Moved from POSIX2 C-language Binding to BASE.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0154 [291] and XSH/TC1-2008/0155 [291] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0121 [806] is applied.

Issue 8

Austin Group Defect 1031 is applied, adding FNM_CASEFOLD and FNM_IGNORECASE.

Austin Group Defect 1287 is applied, changing the description of how <backslash> is handled (if FNM_NOESCAPE is not set) to refer to XCU Section 2.14.1 (on page 2523).

Austin Group Defect 1444 is applied, correcting cross-references to *wordexp*( ).

NAME

fopen — open a stream

SYNOPSIS

```
#include <stdio.h>

FILE *fopen(const char *restrict pathname, const char *restrict mode);
```

DESCRIPTION

CX Except for the ``exclusive access'' requirement (see below), the functionality described on this reference page is aligned with the ISO C standard. Any other conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard for all *fopen*( ) functionality except in relation to ``exclusive access''.

The *fopen*( ) function shall open the file whose pathname is the string pointed to by *pathname*, and associates a stream with it.

The *mode* argument points to a character string. The behavior is unspecified if any character occurs more than once in the string. If the string begins with one of the following characters, then the file shall be opened in the indicated mode. Otherwise, the behavior is undefined.

`'r'` Open file for reading.

`'w'` Truncate to zero length or create file for writing.

`'a'` Append; open or create file for writing at end-of-file.

CX The remainder of the string can contain any of the following characters, in any order, and further affect how the file is opened:

CX `'b'` This character shall have no effect, but is allowed for ISO C standard conformance.

CX `'e'` The underlying file descriptor shall have the FD_CLOEXEC flag atomically set.

CX `'x'` If the first character of *mode* is `'w'` or `'a'`, then the function shall fail if the file already exists, or cannot be created; if the file does not exist and can be created, it shall be created with CX an implementation-defined form of exclusive (also known as non-shared) access, CX if supported by the underlying file system, provided the resulting file permissions are the same as they would be without the `'x'` modifier. If the first character of mode is `'r'`, the effect is implementation-defined.

Note: The ISO C standard requires exclusive access ``to the extent that the underlying file system supports exclusive access'', but does not define what it means by this. Taken at face value—that systems must do whatever they are capable of, at the file system level, in order to exclude access by others—this would require POSIX.1 systems to set the file permissions in a way that prevents access by other users and groups. Consequently, this volume of POSIX.1-2024 does not defer to the ISO C standard as regards the ``exclusive access'' requirement.

`'+'` The file shall be opened for update (both reading and writing), rather than just reading or just writing.

Opening a file with read mode (`'r'` as the first character in the *mode* argument) shall fail if the file does not exist or cannot be read.

Opening a file with append mode (`'a'` as the first character in the *mode* argument) shall cause all subsequent writes to the file to be forced to the then current end-of-file, regardless of intervening calls to *fseek*( ).

When a file is opened with update mode (`'+'` in the *mode* argument), both input and output can be performed on the associated stream. However, the application shall ensure that output is not directly followed by input without an intervening call to *fflush*( ) or to a file positioning function

(*fseek*( ), *fsetpos*( ), or *rewind*( )), and input is not directly followed by output without an intervening call to a file positioning function, unless the input operation encounters end-of-file.

When opened, a stream is fully buffered if and only if it can be determined not to refer to an interactive device. The error and end-of-file indicators for the stream shall be cleared.

CX If the first character in *mode* is `'w'` or `'a'` and the file did not previously exist, upon successful completion, *fopen*( ) shall mark for update the last data access, last data modification, and last file status change timestamps of the file and the last file status change and last data modification timestamps of the parent directory.

If the first character in *mode* is `'w'` or `'a'` and the file did not previously exist, the *fopen*( ) function shall create a file as if it called the *open*( ) function with a value appropriate for the *path* argument interpreted from *pathname*, a value for the *oflag* argument as specified below, and a value of S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH for the third argument.

If the first character in *mode* is `'w'` and the file did previously exist, upon successful completion, *fopen*( ) shall mark for update the last data modification and last file status change timestamps of the file.

XSI After a successful call to the *fopen*( ) function, the orientation of the stream shall be cleared, the encoding rule shall be cleared, and the associated **mbstate_t** object shall be set to describe an initial conversion state.

CX The file descriptor associated with the opened stream shall be allocated and opened as if by a call to *open*( ) using the following flags:

***fopen*() Mode First Character**	***fopen*() Mode Includes `'+'`**	**Initial *open*() Flags**
`'r'`	no	O_RDONLY
`'w'`	no	O_WRONLY|O_CREAT|O_TRUNC
`'a'`	no	O_WRONLY|O_CREAT|O_APPEND
`'r'`	yes	O_RDWR
`'w'`	yes	O_RDWR|O_CREAT|O_TRUNC
`'a'`	yes	O_RDWR|O_CREAT|O_APPEND

If, and only if, the `'e'` *mode* string character is specified, the O_CLOEXEC flag shall be OR'ed into the initial *open*( ) flags specified in the above table.

If, and only if, the `'x'` *mode* string character is specified together with either `'w'` or `'a'`, the O_EXCL flag shall be OR'ed into the initial *open*( ) flags specified in the above table.

If *mode* includes `'x'` and the underlying file system supports exclusive access (see above) enabled by the use of implementation-specific flags to *open*( ), then the behavior shall be as if those flags are also included.

When using *mode* strings specified by this standard, the implementation shall behave as if no other flags had been passed to *open*( ).

RETURN VALUE

Upon successful completion, *fopen*( ) shall return a pointer to the object controlling the stream.
CX Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

33279 **ERRORS**

33280 The *fopen*( ) function shall fail if:

33281 CX [EACCES] Search permission is denied on a component of the path prefix, or the file
33282 exists and the permissions specified by *mode* are denied, or the file does not
33283 exist and write permission is denied for the parent directory of the file to be
33284 created.

33285 CX [EEXIST] The *mode* argument begins with *w* or *a* and includes *x*, but the file already
33286 exists.

33287 CX [EILSEQ] The *mode* argument begins with *w* or *a*, the file did not previously exist, and
33288 the last pathname component is not a portable filename and cannot be created
33289 in the target directory.

33290 CX [EINTR] A signal was caught during *fopen*( ).

33291 CX [EISDIR] The named file is a directory and *mode* requires write access.

33292 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *pathname*
33293 argument.

33294 CX [EMFILE] All file descriptors available to the process are currently open.

33295 CX [EMFILE] {STREAM_MAX} streams are currently open in the calling process.

33296 CX [ENAMETOOLONG]
33297 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
33298 symbolic link produced an intermediate result with a length that exceeds
33299 {PATH_MAX}.

33300 CX [ENFILE] The maximum allowable number of files is currently open in the system.

33301 CX [ENOENT] The *mode* string begins with `'r'` and a component of *pathname* does not name
33302 an existing file, or *mode* begins with `'w'` or `'a'` and a component of the path
33303 prefix of *pathname* does not name an existing file, or *pathname* is an empty
33304 string.

33305 CX [ENOENT] or [ENOTDIR]
33306 The *pathname* argument contains at least one non-<slash> character and ends
33307 with one or more trailing <slash> characters. If *pathname* without the trailing
33308 <slash> characters would name an existing file, an [ENOENT] error shall not
33309 occur.

33310 CX [ENOSPC] The directory or file system that would contain the new file cannot be
33311 expanded, the file does not exist, and the file was to be created.

33312 CX [ENOTDIR] A component of the path prefix names an existing file that is neither a
33313 directory nor a symbolic link to a directory, or the *pathname* argument contains
33314 at least one non-<slash> character and ends with one or more trailing <slash>
33315 characters and the last pathname component names an existing file that is
33316 neither a directory nor a symbolic link to a directory.

33317 CX [ENXIO] The named file is a character special or block special file, and the device
33318 associated with this special file does not exist.

33319 CX [EOVERFLOW] The named file is a regular file and the size of the file cannot be represented
33320 correctly in an object of type **off_t**.

CX [EROFS] The named file resides on a read-only file system and *mode* requires write access.

The *fopen*( ) function may fail if:

CX [EINVAL] The value of the *mode* argument is not valid.

CX [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *pathname* argument.

CX [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

CX [ENAMETOOLONG] The length of a component of a pathname is longer than {NAME_MAX}.

CX [ENOMEM] Insufficient storage space is available.

CX [ETXTBSY] The file is a pure procedure (shared text) file that is being executed and *mode* requires write access.

EXAMPLES

Opening a File

The following example tries to open the file named **file** for reading. The *fopen*( ) function returns a file pointer that is used in subsequent *fgets*( ) and *fclose*( ) calls. If the program cannot open the file, it just ignores it.

```
#include <stdio.h>
...
FILE *fp;
...
void rgrep(const char *file)
{
...
    if ((fp = fopen(file, "r")) == NULL)
        return;
...
}
```

APPLICATION USAGE

If an application needs to create a file in a way that fails if the file already exists, and either requires that it does not have exclusive access to the file or does not need exclusive access, it should use *open*( ) with the O_CREAT and O_EXCL flags instead of using *fopen*( ) with an `'x'` in the mode. A stream can then be created, if needed, by calling *fdopen*( ) on the file descriptor returned by *open*( ).

RATIONALE

The `'e'` mode character is provided as a convenience to avoid a data race in multi-threaded applications. Without it, a file descriptor is leaked into a child process created by one thread in the window between another thread creating a file descriptor with *fopen*( ) and then using *fileno*( ) and *fcntl*( ) to set the FD_CLOEXEC flag. It is also possible to avoid the race by using *open*( ) with O_CLOEXEC followed by *fdopen*( ), however, there is no safe alternative for the *freopen*( ) function, and consistency dictates that the `'e'` modifier should be standardized for all functions that accept *mode* strings.

The ISO C standard only recognizes the `'+'`, `'b'`, and `'x'` characters in certain positions of the *mode* string, leaving other arrangements as unspecified, and only permits `'x'` in *mode* strings

beginning with `'w'`. This standard specifically requires support for all characters other than the first in the *mode* string to be recognized in any order. Thus, `"wxe"` and `"wex"` behave the same, and while `"wx+"` is unspecified in the ISO C standard, this standard requires it to have the same behavior as `"w+x"`. This standard also requires that `'x'` work for *mode* strings beginning with `'a'`, as well as having implementation-defined behavior for *mode* strings beginning with `'r'`. Therefore, while *open*( ) has undefined behavior if O_EXCL is specified without O_CREAT, the same is not true of *fopen*( ).

When `'x'` is in mode, the ISO C standard requires that the file is created with exclusive access to the extent that the underlying system supports exclusive access. Although POSIX.1 does not specify any method of enabling exclusive access, it allows for the existence of an implementation-specific flag, or flags, that enable it. Note that they should be file creation flags if a file is being created, not file access mode flags (that is, ones that are included in O_ACCMODE) or file status flags, so that they do not affect the value returned by *fcntl*( ) with F_GETFL. On implementations that have such flags, if support for them is file system dependent and exclusive access is requested when using *fopen*( ) to create a file on a file system that does not support it, the flags must not be used if they would cause *fopen*( ) to fail.

Some implementations support mandatory file locking as a means of enabling exclusive access to a file. Locks are set in the normal way, but instead of only preventing others from setting conflicting locks they prevent others from accessing the contents of the locked part of the file in a way that conflicts with the lock. However, unless the implementation has a way of setting a whole-file write lock on file creation, this does not satisfy the requirement in the ISO C standard that the file is ``created with exclusive access to the extent that the underlying system supports exclusive access''. (Having *fopen*( ) create the file and set a lock on the file as two separate operations is not the same, and it would introduce a race condition whereby another process could open the file and write to it (or set a lock) in between the two operations.) However, on all implementations that support mandatory file locking, its use is discouraged; therefore, it is recommended that implementations which support mandatory file locking do not add a means of creating a file with a whole-file exclusive lock set, so that *fopen*( ) is not required to enable mandatory file locking in order to conform to the ISO C standard. An implementation that has a means of creating a file with a whole-file exclusive lock set would need to provide a way to change the behavior of *fopen*( ) depending on whether the calling process is executing in a POSIX.1 conforming environment or an ISO C conforming environment.

The typical implementation-defined behavior for mode `"rx"` is to ignore the `'x'`, but the standard developers did not wish to mandate this behavior. For example, an implementation could allow shared access for reading; that is, disallow a file that has been opened this way from also being opened for writing.

Implementations are encouraged to have *fopen*( ) and *freopen*( ) report an [EILSEQ] error if *mode* begins with `'w'` or `'a'`, the file did not previously exist, and the last component of *pathname* contains any bytes that have the encoded value of a <newline> character.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *creat*( ), *fclose*( ), *fdopen*( ), *fmemopen*( ), *freopen*( ), *open_memstream*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files.
- In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support large files.
- The [ELOOP] mandatory error condition is added.
- The [EINVAL], [EMFILE], [ENAMETOOLONG], [ENOMEM], and [ETXTBSY] optional error conditions are added.

The normative text is updated to avoid use of the term ``must'' for application requirements.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The prototype for *fopen*( ) is updated.
- The DESCRIPTION is updated to note that if the argument *mode* points to a string other than those listed, then the behavior is undefined.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #025 is applied, clarifying the file creation mode.

Austin Group Interpretation 1003.1-2001 #143 is applied.

Austin Group Interpretation 1003.1-2001 #159 is applied, clarifying requirements for the flags set on the open file description.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-149 is applied, changing the {STREAM_MAX} [EMFILE] error condition from a ``may fail'' to a ``shall fail''.

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0156 [291,433], XSH/TC1-2008/0157 [146,433], XSH/TC1-2008/0158 [324], and XSH/TC1-2008/0159 [14] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0122 [822] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 293 is applied, adding the [EILSEQ] error.

Austin Group Defects 411 and 1524 are applied, adding the `'e'` and `'x'` *mode* string characters.

Austin Group Defect 1200 is applied, correcting the argument name in the [ELOOP] errors.

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899: 2018 standard.

33451 **NAME**

33452 fork — create a new process

33453 **SYNOPSIS**

33454
```
#include <unistd.h>
```

33455
```
pid_t fork(void);
```
33456
```
pid_t _Fork(void);
```

33457 **DESCRIPTION**

33458 The *fork*( ) function shall create a new process. The new process (child process) shall be an exact
33459 copy of the calling process (parent process) except as detailed below:

33460 • The child process shall have a unique process ID.

33461 • The child process ID also shall not match any active process group ID.

33462 • The child process shall have a different parent process ID, which shall be the process ID of
33463 the calling process.

33464 • The child process shall have its own copy of the parent's file descriptors, except for those
33465 whose FD_CLOFORK flag is set (see *fcntl*( )). Each of the child's file descriptors shall refer
33466 to the same open file description with the corresponding file descriptor of the parent.

33467 • The child process shall have its own copy of the parent's open directory streams. Each
33468 open directory stream in the child process may share directory stream positioning with the
33469 corresponding directory stream of the parent.

33470 • The child process shall have its own copy of the parent's message catalog descriptors.

33471 • The child process values of *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime* shall be set to
33472 0.

33473 • The time left until an alarm clock signal shall be reset to zero, and the alarm, if any, shall be
33474 canceled; see *alarm*( ).

33475 XSI • All *semadj* values shall be cleared.

33476 • Process-owned file locks set by the parent process shall not be inherited by the child
33477 process.

33478 • The set of signals pending for the child process shall be initialized to the empty set.

33479 XSI • Interval timers shall be reset in the child process.

33480 • Any semaphores that are open in the parent process shall also be open in the child process.

33481 ML • The child process shall not inherit any address space memory locks established by the
33482 parent process via calls to *mlockall*( ) or *mlock*( ).

33483 • Memory mappings created in the parent shall be retained in the child process.
33484 MAP_PRIVATE mappings inherited from the parent shall also be MAP_PRIVATE
33485 mappings in the child, and any modifications to the data in these mappings made by the
33486 parent prior to calling *fork*( ) shall be visible to the child. Any modifications to the data in
33487 MAP_PRIVATE mappings made by the parent after *fork*( ) returns shall be visible only to
33488 the parent. Modifications to the data in MAP_PRIVATE mappings made by the child shall
33489 be visible only to the child.

33490 PS • For the SCHED_FIFO and SCHED_RR scheduling policies, the child process shall inherit
33491 the policy and priority settings of the parent process during a *fork*( ) function. For other
33492 scheduling policies, the policy and priority settings on *fork*( ) are implementation-defined.

- Per-process timers created by the parent shall not be inherited by the child process.
- MSG The child process shall have its own copy of the message queue descriptors of the parent. Each of the message descriptors of the child shall refer to the same open message queue description as the corresponding message descriptor of the parent.
- No asynchronous input or asynchronous output operations shall be inherited by the child process. Any use of asynchronous control blocks created by the parent produces undefined behavior.
- A process shall be created with a single thread. If a multi-threaded process calls *fork*( ), the new process shall contain a replica of the calling thread and its entire address space, possibly including the states of mutexes and other resources. Consequently, the application shall ensure that the child process only executes async-signal-safe operations until such time as one of the *exec* functions is successful.
- Any locks held by any thread in the calling process that have been set to be process-shared shall not be held by the child process. For locks held by any thread in the calling process that have not been set to be process-shared, any attempt by the child process to perform any operation on the lock results in undefined behavior (regardless of whether the calling process is single-threaded or multi-threaded).
- CPT The initial value of the CPU-time clock of the child process shall be set to zero.
- TCT The initial value of the CPU-time clock of the single thread of the child process shall be set to zero.

All other process characteristics defined by POSIX.1-2024 shall be the same in the parent and child processes. The inheritance of process characteristics not defined by POSIX.1-2024 is unspecified by POSIX.1-2024.

After *fork*( ), both the parent and the child processes shall be capable of executing independently before either one terminates.

The *_Fork*( ) function shall be equivalent to *fork*( ), except that fork handlers established by means of the *pthread_atfork*( ) function shall not be called and *_Fork*( ) shall be async-signal-safe.

RETURN VALUE

Upon successful completion, *fork*( ) shall return 0 to the child process and shall return the process ID of the child process to the parent process. Both processes shall continue to execute from the *fork*( ) function. Otherwise, –1 shall be returned to the parent process, no child process shall be created, and *errno* shall be set to indicate the error.

ERRORS

These functions shall fail if:

[EAGAIN] The system lacked the necessary resources to create another process, or the system-imposed limit on the total number of processes under execution system-wide or by a single user {CHILD_MAX} would be exceeded.

These functions may fail if:

[ENOMEM] Insufficient storage space is available.

EXAMPLES

None.

APPLICATION USAGE

When a multi-threaded process calls *fork*( ) or *_Fork*( ), there is no guarantee that thread-specific memory, such as stacks or thread-local storage, associated with threads in the parent other than the calling thread will still be available in the child. This is because threads in the parent other than the calling thread do not exist in the child. Consequently, the implementation of *fork*( ) or *_Fork*( ) could remove that memory from the address space in the child, or reuse it for other purposes before returning or (in the case of *fork*( )) calling any of the fork handlers registered by *pthread_atfork*( ). Therefore, applications should avoid using any pointers to thread-specific memory in the child that were passed to the calling thread from other threads in the parent.

RATIONALE

Many historical implementations have timing windows where a signal sent to a process group (for example, an interactive SIGINT) just prior to or during execution of *fork*( ) is delivered to the parent following the *fork*( ) but not to the child because the *fork*( ) code clears the child's set of pending signals. This volume of POSIX.1-2024 does not require, or even permit, this behavior. However, it is pragmatic to expect that problems of this nature may continue to exist in implementations that appear to conform to this volume of POSIX.1-2024 and pass available verification suites. This behavior is only a consequence of the implementation failing to make the interval between signal generation and delivery totally invisible. From the application's perspective, a *fork*( ) call should appear atomic. A signal that is generated prior to the *fork*( ) should be delivered prior to the *fork*( ). A signal sent to the process group after the *fork*( ) should be delivered to both parent and child. The implementation may actually initialize internal data structures corresponding to the child's set of pending signals to include signals sent to the process group during the *fork*( ). Since the *fork*( ) call can be considered as atomic from the application's perspective, the set would be initialized as empty and such signals would have arrived after the *fork*( ); see also **<signal.h>**.

One approach that has been suggested to address the problem of signal inheritance across *fork*( ) is to add an [EINTR] error, which would be returned when a signal is detected during the call. While this is preferable to losing signals, it was not considered an optimal solution. Although it is not recommended for this purpose, such an error would be an allowable extension for an implementation.

The [ENOMEM] error value is reserved for those implementations that detect and distinguish such a condition. This condition occurs when an implementation detects that there is not enough memory to create the process. This is intended to be returned when [EAGAIN] is inappropriate because there can never be enough memory (either primary or secondary storage) to perform the operation. Since *fork*( ) duplicates an existing process, this must be a condition where there is sufficient memory for one such process, but not for two. Many historical implementations actually return [ENOMEM] due to temporary lack of memory, a case that is not generally distinct from [EAGAIN] from the perspective of a conforming application.

Part of the reason for including the optional error [ENOMEM] is because the SVID specifies it and it should be reserved for the error condition specified there. The condition is not applicable on many implementations.

IEEE Std 1003.1-1988 neglected to require concurrent execution of the parent and child of *fork*( ). A system that single-threads processes was clearly not intended and is considered an unacceptable ``toy implementation'' of this volume of POSIX.1-2024. The only objection anticipated to the phrase ``executing independently'' is testability, but this assertion should be testable. Such tests require that both the parent and child can block on a detectable action of the other, such as a write to a pipe or a signal. An interactive exchange of such actions should be

possible for the system to conform to the intent of this volume of POSIX.1-2024.

The [EAGAIN] error exists to warn applications that such a condition might occur. Whether it occurs or not is not in any practical sense under the control of the application because the condition is usually a consequence of the user's use of the system, not of the application's code. Thus, no application can or should rely upon its occurrence under any circumstances, nor should the exact semantics of what concept of ``user'' is used be of concern to the application developer. Validation writers should be cognizant of this limitation.

There are two reasons why POSIX programmers call *fork*( ). One reason is to create a new thread of control within the same program (which was originally only possible in POSIX by creating a new process); the other is to create a new process running a different program. In the latter case, the call to *fork*( ) is soon followed by a call to one of the *exec* functions.

The general problem with making *fork*( ) work in a multi-threaded world is what to do with all of the threads. There are two alternatives. One is to copy all of the threads into the new process. This causes the programmer or implementation to deal with threads that are suspended on system calls or that might be about to execute system calls that should not be executed in the new process. The other alternative is to copy only the thread that calls *fork*( ). This creates the difficulty that the state of process-local resources is usually held in process memory. If a thread that is not calling *fork*( ) holds a resource, that resource is never released in the child process because the thread whose job it is to release the resource does not exist in the child process.

When a programmer is writing a multi-threaded program, the first described use of *fork*( ), creating new threads in the same program, is provided by the *pthread_create*( ) function. The *fork*( ) function is thus used only to run new programs, and the effects of calling functions that require certain resources between the call to *fork*( ) and the call to an *exec* function are undefined.

The addition of the *forkall*( ) function to the standard was considered and rejected. The *forkall*( ) function lets all the threads in the parent be duplicated in the child. This essentially duplicates the state of the parent in the child. This allows threads in the child to continue processing and allows locks and the state to be preserved without explicit *pthread_atfork*( ) code. The calling process has to ensure that the threads processing state that is shared between the parent and child (that is, file descriptors or MAP_SHARED memory) behaves properly after *forkall*( ). For example, if a thread is reading a file descriptor in the parent when *forkall*( ) is called, then two threads (one in the parent and one in the child) are reading the file descriptor after the *forkall*( ). If this is not desired behavior, the parent process has to synchronize with such threads before calling *forkall*( ).

When *forkall*( ) is called, threads, other than the calling thread, that are in functions that can return with an [EINTR] error may have those functions return [EINTR] if the implementation cannot ensure that the function behaves correctly in the parent and child. In particular, *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), and *pthread_cond_wait*( ) need to return in order to ensure that the condition has not changed. These functions can be awakened by a spurious condition wakeup rather than returning [EINTR].

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *exec*, *fcntl*( ), *pthread_atfork*( ), *semop*( ), *signal*( ), *times*( )

XBD Section 4.15.2 (on page 104), **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is changed for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effect of *fork*( ) on a pending alarm call in the child process is clarified.

The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/17 is applied, adding text to the DESCRIPTION and RATIONALE relating to fork handlers registered by the *pthread_atfork*( ) function and async-signal safety.

Issue 7

Austin Group Interpretation 1003.1-2001 #080 is applied, clarifying the status of asynchronous input and asynchronous output operations and asynchronous control lists in the DESCRIPTION.

Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, Timers, and Threads options is moved to the Base.

Functionality relating to message catalog descriptors is moved from the XSI option to the Base.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0123 [858] is applied.

Issue 8

Austin Group Defects 62, 1361, and 1383 are applied, adding the *_Fork*( ) function and removing the requirement for *fork*( ) to be async-signal-safe.

Austin Group Defect 768 is applied, adding OFD-owned file locks.

Austin Group Defect 1112 is applied, clarifying the requirements for a child of a multi-threaded process and for process-shared and non-process-shared locks held by any thread in the calling process.

Austin Group Defect 1114 is applied, changing the APPLICATION USAGE section.

Austin Group Defect 1216 is applied, adding *pthread_cond_clockwait*( ).

Austin Group Defect 1318 is applied, adding FD_CLOFORK.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

NAME

fpathconf, pathconf — get configurable pathname variables

SYNOPSIS

```
#include <unistd.h>

long fpathconf(int fildes, int name);
long pathconf(const char *path, int name);
```

DESCRIPTION

The *fpathconf*( ) and *pathconf*( ) functions shall determine the current value of a configurable limit or option (*variable*) that is associated with a file or directory.

For *pathconf*( ), the *path* argument points to the pathname of a file or directory.

For *fpathconf*( ), the *fildes* argument is an open file descriptor.

The *name* argument represents the variable to be queried relative to that file or directory. Implementations shall support all of the variables listed in the following table and may support others. The variables in the following table come from **<limits.h>** or **<unistd.h>** and the symbolic constants, defined in **<unistd.h>**, are the corresponding values used for *name*.

Variable	Value of *name*	Requirements
{FILESIZEBITS}	_PC_FILESIZEBITS	4,7
{LINK_MAX}	_PC_LINK_MAX	1
{MAX_CANON}	_PC_MAX_CANON	2
{MAX_INPUT}	_PC_MAX_INPUT	2
{NAME_MAX}	_PC_NAME_MAX	3,4
{PATH_MAX}	_PC_PATH_MAX	4,5
{PIPE_BUF}	_PC_PIPE_BUF	6
{POSIX2_SYMLINKS}	_PC_2_SYMLINKS	4
{POSIX_ALLOC_SIZE_MIN}	_PC_ALLOC_SIZE_MIN	10
{POSIX_REC_INCR_XFER_SIZE}	_PC_REC_INCR_XFER_SIZE	10
{POSIX_REC_MAX_XFER_SIZE}	_PC_REC_MAX_XFER_SIZE	10
{POSIX_REC_MIN_XFER_SIZE}	_PC_REC_MIN_XFER_SIZE	10
{POSIX_REC_XFER_ALIGN}	_PC_REC_XFER_ALIGN	10
{SYMLINK_MAX}	_PC_SYMLINK_MAX	4,9
{TEXTDOMAIN_MAX}	_PC_TEXTDOMAIN_MAX	3,4
_POSIX_CHOWN_RESTRICTED	_PC_CHOWN_RESTRICTED	7
_POSIX_NO_TRUNC	_PC_NO_TRUNC	3,4
_POSIX_VDISABLE	_PC_VDISABLE	2
_POSIX_ASYNC_IO	_PC_ASYNC_IO	8
_POSIX_FALLOC	_PC_FALLOC	8
_POSIX_PRIO_IO	_PC_PRIO_IO	8
_POSIX_SYNC_IO	_PC_SYNC_IO	8
_POSIX_TIMESTAMP_RESOLUTION	_PC_TIMESTAMP_RESOLUTION	1

Requirements

1. If *path* or *fildes* refers to a directory, the value returned shall apply to the directory itself.
2. If *path* or *fildes* does not refer to a terminal file, it is unspecified whether an implementation supports an association of the variable name with the specified file.
3. If *path* or *fildes* refers to a directory, the value returned shall apply to filenames within the directory.
4. If *path* or *fildes* does not refer to a directory, it is unspecified whether an implementation supports an association of the variable name with the specified file.
5. If *path* or *fildes* refers to a directory, the value returned shall be the maximum length of a relative pathname that would not cross any mount points when the specified directory is the working directory.
6. If *path* refers to a FIFO, or *fildes* refers to a pipe or FIFO, the value returned shall apply to the referenced object. If *path* or *fildes* refers to a directory, the value returned shall apply to any FIFO that exists or can be created within the directory. If *path* or *fildes* refers to any other type of file, it is unspecified whether an implementation supports an association of the variable name with the specified file.
7. If *path* or *fildes* refers to a directory, the value returned shall apply to any files, other than directories, that exist or can be created within the directory.
8. If *path* or *fildes* refers to a directory, it is unspecified whether an implementation supports an association of the variable name with the specified file.
9. If *path* or *fildes* refers to a directory, the value returned shall be the maximum length of the string that a symbolic link in that directory can contain.
10. If *path* or *fildes* des does not refer to a regular file, it is unspecified whether an implementation supports an association of the variable name with the specified file. If an implementation supports such an association for other than a regular file, the value returned is unspecified.

RETURN VALUE

If *name* is an invalid value, both *pathconf*( ) and *fpathconf*( ) shall return –1 and set *errno* to indicate the error.

If the variable corresponding to *name* is described in **<limits.h>** as a maximum or minimum value and the variable has no limit for the path or file descriptor, both *pathconf*( ) and *fpathconf*( ) shall return –1 without changing *errno*. Note that indefinite limits do not imply infinite limits; see **<limits.h>**.

If the implementation needs to use *path* to determine the value of *name* and the implementation does not support the association of *name* with the file specified by *path*, or if the process did not have appropriate privileges to query the file specified by *path*, or *path* does not exist, *pathconf*( ) shall return –1 and set *errno* to indicate the error.

If the implementation needs to use *fildes* to determine the value of *name* and the implementation does not support the association of *name* with the file specified by *fildes*, or if *fildes* is an invalid file descriptor, *fpathconf*( ) shall return –1 and set *errno* to indicate the error.

Otherwise, *pathconf*( ) or *fpathconf*( ) shall return the current variable value for the file or directory without changing *errno*. The value returned shall not be more restrictive than the corresponding value available to the application when it was compiled with the implementation's **<limits.h>** or **<unistd.h>**.

33744 If the variable corresponding to *name* is dependent on an unsupported option, the results are
33745 unspecified.

33746 **ERRORS**

33747 The *pathconf*( ) function shall fail if:

33748 [EINVAL] The value of *name* is not valid.

33749 [EOVERFLOW] The value of *name* is _PC_TIMESTAMP_RESOLUTION and the resolution is
33750 larger than {LONG_MAX}.

33751 The *pathconf*( ) function may fail if:

33752 [EACCES] Search permission is denied for a component of the path prefix.

33753 [EINVAL] The implementation does not support an association of the variable *name* with
33754 the specified file.

33755 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
33756 argument.

33757 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
33758 resolution of the *path* argument.

33759 [ENAMETOOLONG]
33760 The length of a component of a pathname is longer than {NAME_MAX}.

33761 [ENAMETOOLONG]
33762 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
33763 symbolic link produced an intermediate result with a length that exceeds
33764 {PATH_MAX}.

33765 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

33766 [ENOTDIR] A component of the path prefix names an existing file that is neither a
33767 directory nor a symbolic link to a directory, or the *path* argument contains at
33768 least one non-<slash> character and ends with one or more trailing <slash>
33769 characters and the last pathname component names an existing file that is
33770 neither a directory nor a symbolic link to a directory.

33771 The *fpathconf*( ) function shall fail if:

33772 [EINVAL] The value of *name* is not valid.

33773 [EOVERFLOW] The value of *name* is _PC_TIMESTAMP_RESOLUTION and the resolution is
33774 larger than {LONG_MAX}.

33775 The *fpathconf*( ) function may fail if:

33776 [EBADF] The *fildes* argument is not a valid file descriptor.

33777 [EINVAL] The implementation does not support an association of the variable *name* with
33778 the specified file.

33779 **EXAMPLES**

33780 None.

33781 **APPLICATION USAGE**

33782 Application developers should check whether an option, such as _POSIX_ADVISORY_INFO, is
33783 supported prior to obtaining and using values for related variables such as
33784 {POSIX_ALLOC_SIZE_MIN}.

33785 **RATIONALE**

33786 The *pathconf*( ) function was proposed immediately after the *sysconf*( ) function when it was
33787 realized that some configurable values may differ across file system, directory, or device
33788 boundaries.

33789 For example, {NAME_MAX} frequently changes between System V and BSD-based file systems;
33790 System V uses a maximum of 14, BSD 255. On an implementation that provides both types of file
33791 systems, an application would be forced to limit all pathname components to 14 bytes, as this
33792 would be the value specified in **<limits.h>** on such a system.

33793 Therefore, various useful values can be queried on any pathname or file descriptor, assuming
33794 that appropriate privileges are in place.

33795 The value returned for the variable {PATH_MAX} indicates the longest relative pathname that
33796 could be given if the specified directory is the current working directory of the process. A
33797 process may not always be able to generate a name that long and use it if a subdirectory in the
33798 pathname crosses into a more restrictive file system. Note that implementations are allowed to
33799 accept pathnames longer than {PATH_MAX} bytes long, but are not allowed to return
33800 pathnames longer than this unless the user specifies a larger buffer using a function that
33801 provides a buffer size argument.

33802 The value returned for the variable _POSIX_CHOWN_RESTRICTED also applies to directories
33803 that do not have file systems mounted on them. The value may change when crossing a mount
33804 point, so applications that need to know should check for each directory. (An even easier check
33805 is to try the *chown*( ) function and look for an error in case it happens.)

33806 Unlike the values returned by *sysconf*( ), the pathname-oriented variables are potentially more
33807 volatile and are not guaranteed to remain constant throughout the lifetime of the process. For
33808 example, in between two calls to *pathconf*( ), the file system in question may have been
33809 unmounted and remounted with different characteristics.

33810 Also note that most of the errors are optional. If one of the variables always has the same value
33811 on an implementation, the implementation need not look at *path* or *fildes* to return that value and
33812 is, therefore, not required to detect any of the errors except the meaning of [EINVAL] that
33813 indicates that the value of *name* is not valid for that variable, and the [EOVERFLOW] error that
33814 indicates the value to be returned is larger than {LONG_MAX}.

33815 If the value of any of the limits is unspecified (logically infinite), they will not be defined in
33816 **<limits.h>** and the *pathconf*( ) and *fpathconf*( ) functions return –1 without changing *errno*. This
33817 can be distinguished from the case of giving an unrecognized *name* argument because *errno* is set
33818 to [EINVAL] in this case.

33819 Since –1 is a valid return value for the *pathconf*( ) and *fpathconf*( ) functions, applications should
33820 set *errno* to zero before calling them and check *errno* only if the return value is –1.

33821 For the case of {SYMLINK_MAX}, since both *pathconf*( ) and *open*( ) follow symbolic links, there
33822 is no way that *path* or *fildes* could refer to a symbolic link.

33823 It was the intention of IEEE Std 1003.1d-1999 that the following variables:

{POSIX_ALLOC_SIZE_MIN}
{POSIX_REC_INCR_XFER_SIZE}
{POSIX_REC_MAX_XFER_SIZE}
{POSIX_REC_MIN_XFER_SIZE}
{POSIX_REC_XFER_ALIGN}

only applied to regular files, but Note 10 also permits implementation of the advisory semantics on other file types unique to an implementation (for example, a character special device).

The [EOVERFLOW] error for _PC_TIMESTAMP_RESOLUTION cannot occur on POSIX-compliant file systems because POSIX requires a timestamp resolution no larger than one second. Even on 32-bit systems, this can be represented without overflow.

FUTURE DIRECTIONS

None.

SEE ALSO

chown( ), *confstr( )*, *sysconf( )*

XBD **<limits.h>**, **<unistd.h>**

XCU *getconf*

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

Large File Summit extensions are added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to include {FILESIZEBITS}.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The _PC_SYMLINK_MAX entry is added to the table in the DESCRIPTION.

The following *pathconf( )* variables and their associated names are added for alignment with IEEE Std 1003.1d-1999:

{POSIX_ALLOC_SIZE_MIN}
{POSIX_REC_INCR_XFER_SIZE}
{POSIX_REC_MAX_XFER_SIZE}
{POSIX_REC_MIN_XFER_SIZE}
{POSIX_REC_XFER_ALIGN}

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/18 is applied, changing the fourth paragraph of the DESCRIPTION and removing shading and margin markers from the table. This change is needed since implementations are required to support all of these symbols.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/34 is applied, adding the table entry for POSIX2_SYMLINKS in the DESCRIPTION.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/35 is applied, updating the DESCRIPTION and RATIONALE sections to clarify behavior for the following variables:

{POSIX_ALLOC_SIZE_MIN}
{POSIX_REC_INCR_XFER_SIZE}
{POSIX_REC_MAX_XFER_SIZE}
{POSIX_REC_MIN_XFER_SIZE}
{POSIX_REC_XFER_ALIGN}

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/36 is applied, updating the RETURN VALUE and APPLICATION USAGE sections to state that the results are unspecified if a variable is dependent on an unsupported option, and advising application developers to check for supported options prior to obtaining and using such values.

Issue 7

Austin Group Interpretations 1003.1-2001 #143 and #160 are applied.

Changes are made related to support for finegrained timestamps.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0160 [256,428], XSH/TC1-2008/0161 [256,428], and XSH/TC1-2008/0162 [324] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0124 [651] and XSH/TC2-2008/0125 [651] are applied.

Issue 8

Austin Group Defect 687 is applied, adding _POSIX_FALLOC and _PC_FALLOC.

Austin Group Defect 1122 is applied, adding {TEXTDOMAIN_MAX}.

33888 **NAME**

33889 fpclassify — classify real floating type

33890 **SYNOPSIS**

33891 ```
#include <math.h>
```

33892 ```
int fpclassify(real-floating x);
```

33893 **DESCRIPTION**

33894 CX The functionality described on this reference page is aligned with the ISO C standard. Any
33895 conflict between the requirements described here and the ISO C standard is unintentional. This
33896 volume of POSIX.1-2024 defers to the ISO C standard.

33897 The *fpclassify*( ) macro shall classify its argument value as NaN, infinite, normal, subnormal,
33898 zero, or into another implementation-defined category. First, an argument represented in a
33899 format wider than its semantic type is converted to its semantic type. Then classification is based
33900 on the type of the argument.

33901 **RETURN VALUE**

33902 The *fpclassify*( ) macro shall return the value of the number classification macro appropriate to
33903 the value of its argument.

33904 **ERRORS**

33905 No errors are defined.

33906 **EXAMPLES**

33907 None.

33908 **APPLICATION USAGE**

33909 None.

33910 **RATIONALE**

33911 None.

33912 **FUTURE DIRECTIONS**

33913 None.

33914 **SEE ALSO**

33915 *isfinite*( ), *isinf*( ), *isnan*( ), *isnormal*( ), *signbit*( )

33916 XBD **<math.h>**

33917 **CHANGE HISTORY**

33918 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

33919 **NAME**

33920 asprintf, dprintf, fprintf, printf, snprintf, sprintf — print formatted output

33921 **SYNOPSIS**

33922
```
#include <stdio.h>
```

33923 CX
```
int asprintf(char **restrict ptr, const char *restrict format, ...);
int dprintf(int fildes, const char *restrict format, ...);
int fprintf(FILE *restrict stream, const char *restrict format, ...);
int printf(const char *restrict format, ...);
int snprintf(char *restrict s, size_t n,
    const char *restrict format, ...);
int sprintf(char *restrict s, const char *restrict format, ...);
```

33930 **DESCRIPTION**

33931 CX Except for *asprintf*( ), *dprintf*( ), and the behavior of the `%lc` conversion when passed a null wide character, the functionality described on this reference page is aligned with the ISO C standard. Any other conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard for all *fprintf*( ), *printf*( ), *snprintf*( ), and *sprintf*( ) functionality except in relation to the `%lc` conversion when passed a null wide character.

33937 The *fprintf*( ) function shall place output on the named output *stream*. The *printf*( ) function shall place output on the standard output stream *stdout*. The *sprintf*( ) function shall place output followed by the null byte, `'\0'`, in consecutive bytes starting at **s*; it is the user's responsibility to ensure that enough space is available.

33941 CX The *asprintf*( ) function shall be equivalent to *sprintf*( ), except that the output string shall be written to dynamically allocated memory, allocated as if by a call to *malloc*( ), of sufficient length to hold the resulting string, including a terminating null byte. If the call to *asprintf*( ) is successful, the address of this dynamically allocated string shall be stored in the location referenced by *ptr*.

33946 The *dprintf*( ) function shall be equivalent to the *fprintf*( ) function, except that *dprintf*( ) shall write output to the file associated with the file descriptor specified by the *fildes* argument rather than place output on a stream.

33949 The *snprintf*( ) function shall be equivalent to *sprintf*( ), with the addition of the *n* argument which limits the number of bytes written to the buffer referred to by *s*. If *n* is zero, nothing shall be written and *s* may be a null pointer. Otherwise, output bytes beyond the *n*-1st shall be discarded instead of being written to the array, and a null byte is written at the end of the bytes actually written into the array.

33954 If copying takes place between objects that overlap as a result of a call to *sprintf*( ) or *snprintf*( ), the results are undefined.

33956 Each of these functions converts, formats, and prints its arguments under control of the *format*. The application shall ensure that the format is a character string, beginning and ending in its initial shift state, if any. The format is composed of zero or more directives: *ordinary characters*, which are simply copied to the output stream, and *conversion specifications*, each of which shall result in the fetching of zero or more arguments. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments shall be evaluated but are otherwise ignored.

33963 CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than to the next unused argument. In this case, the conversion specifier character `%` (see below) is replaced by the sequence `"%`*n*`$"`, where *n* is a decimal integer in the range [1,{NL_ARGMAX}],

giving the position of the argument in the argument list. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages (see the EXAMPLES section).

The *format* can contain either numbered argument conversion specifications (that is, those introduced by "%*n*$" and optionally containing the "**m*$" forms of field width and precision), or unnumbered argument conversion specifications (that is, those introduced by the % character and optionally containing the * form of field width and precision), but not both. The only exception to this is that %% can be mixed with the "%*n*$" form. The results of mixing numbered and unnumbered argument specifications in a format string are undefined. When numbered argument specifications are used, specifying the *N*th argument requires that all the leading arguments, from the first to the (*N*–*1*)th, are specified in the format string.

In format strings containing the "%*n*$" form of conversion specification, numbered arguments in the argument list can be referenced from the format string as many times as required.

In format strings containing the % form of conversion specification, each conversion specification uses the first unused argument in the argument list.

CX All forms of the *fprintf*() functions allow for the insertion of a language-dependent radix character in the output string. The radix character is defined in the current locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a <period> ('.').

CX Each conversion specification is introduced by the '%' character or by the character sequence "%*n*$", after which the following appear in sequence:

- Zero or more *flags* (in any order), which modify the meaning of the conversion specification.
- An optional minimum *field width*. If the converted value has fewer bytes than the field width, it shall be padded with <space> characters by default on the left; it shall be padded on the right if the left-adjustment flag ('–'), described below, is given to the field width. CX The field width takes the form of an <asterisk> ('*'), or in conversion specifications introduced by "%*n*$" the "**m*$" string, described below, or a decimal integer.
- An optional *precision* that gives the minimum number of digits to appear for the d, i, o, u, x, and X conversion specifiers; the number of digits to appear after the radix character for the a, A, e, E, f, and F conversion specifiers; the maximum number of significant digits for the g and G conversion specifiers; or the maximum number of bytes to be printed from a string in the s XSI and S conversion specifiers. The precision takes the form of a <period> ('.') CX followed either by an <asterisk> ('*'), or in conversion specifications introduced by "%*n*$" the "**m*$" string, described below, or an optional decimal digit string, where a null digit string is treated as zero. If a precision appears with any other conversion specifier, the behavior is undefined.
- An optional length modifier that specifies the size of the argument.
- A *conversion specifier* character that indicates the type of conversion to be applied.

A field width, or precision, or both, may be indicated by an <asterisk> ('*'). In this case an argument of type **int** supplies the field width or precision. Applications shall ensure that arguments specifying field width, or precision, or both appear in that order before the argument, if any, to be converted. A negative field width is taken as a '–' flag followed by a positive field CX width. A negative precision is taken as if the precision were omitted. In format strings containing conversion specifications introduced by "%*n*$", in addition to being indicated by the decimal digit string, a field width may be indicated by the sequence "**m*$" and precision by the

sequence "`.*`*m*`$`", where *m* is a decimal integer in the range [1,{NL_ARGMAX}] giving the position in the argument list (after the *format* argument) of an integer argument containing the field width or precision, for example:

```
printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

The flag characters and their meanings are:

CX `'` (The <apostrophe>.) The integer portion of the result of a decimal conversion (`%i`, `%d`, `%u`, `%f`, `%F`, `%g`, or `%G`) shall be formatted with thousands' grouping characters. For other conversions the behavior is undefined. The non-monetary grouping character is used.

`-` The result of the conversion shall be left-justified within the field. The conversion is right-justified if this flag is not specified.

`+` The result of a signed conversion shall always begin with a sign (`'+'` or `'-'`). The conversion shall begin with a sign only when a negative value is converted if this flag is not specified.

<space> If the first character of a signed conversion is not a sign or if a signed conversion results in no characters, a <space> shall be prefixed to the result. This means that if the <space> and `'+'` flags both appear, the <space> flag shall be ignored.

`#` Specifies that the value is to be converted to an alternative form. For `o` conversion, it shall increase the precision, if and only if necessary, to force the first digit of the result to be a zero (if the value and precision are both 0, a single 0 is printed). For `x` or `X` conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For `a`, `A`, `e`, `E`, `f`, `F`, `g`, and `G` conversion specifiers, the result shall always contain a radix character, even if no digits follow the radix character. Without this flag, a radix character appears in the result of these conversions only if a digit follows it. For `g` and `G` conversion specifiers, trailing zeros shall *not* be removed from the result as they normally are. For other conversion specifiers, the behavior is undefined.

`0` For `d`, `i`, `o`, `u`, `x`, `X`, `a`, `A`, `e`, `E`, `f`, `F`, `g`, and `G` conversion specifiers, leading zeros (following any indication of sign or base) are used to pad to the field width rather than performing space padding, except when converting an infinity or NaN. If the `'0'` and `'-'` flags both appear, the `'0'` flag is ignored. For `d`, `i`, `o`, `u`, `x`, and `X` conversion
CX specifiers, if a precision is specified, the `'0'` flag shall be ignored. If the `'0'` and <apostrophe> flags both appear, the grouping characters are inserted before zero padding. For other conversions, the behavior is undefined.

The length modifiers and their meanings are:

`hh` Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **signed char** or **unsigned char** argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to **signed char** or **unsigned char** before printing); or that a following `n` conversion specifier applies to a pointer to a **signed char** argument.

`h` Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **short** or **unsigned short** argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to **short** or **unsigned short** before printing); or that a following `n` conversion specifier applies to a pointer to a **short** argument.

34056 l (ell) Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long** or
34057 **unsigned long** argument; that a following n conversion specifier applies to a pointer to
34058 a **long** argument; that a following c conversion specifier applies to a **wint_t** argument;
34059 that a following s conversion specifier applies to a pointer to a **wchar_t** argument; or
34060 has no effect on a following a, A, e, E, f, F, g, or G conversion specifier.

34061 ll (ell-ell)
34062 Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long long**
34063 or **unsigned long long** argument; or that a following n conversion specifier applies to a
34064 pointer to a **long long** argument.

34065 j Specifies that a following d, i, o, u, x, or X conversion specifier applies to an **intmax_t**
34066 or **uintmax_t** argument; or that a following n conversion specifier applies to a pointer
34067 to an **intmax_t** argument.

34068 z Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **size_t** or the
34069 corresponding signed integer type argument; or that a following n conversion specifier
34070 applies to a pointer to a signed integer type corresponding to a **size_t** argument.

34071 t Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **ptrdiff_t** or
34072 the corresponding **unsigned** type argument; or that a following n conversion specifier
34073 applies to a pointer to a **ptrdiff_t** argument.

34074 L Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to a **long**
34075 **double** argument.

34076 If a length modifier appears with any conversion specifier other than as specified above, the
34077 behavior is undefined.

34078 The conversion specifiers and their meanings are:

34079 d, i The **int** argument shall be converted to a signed decimal in the style "[−]*dddd*". The
34080 precision specifies the minimum number of digits to appear; if the value being
34081 converted can be represented in fewer digits, it shall be expanded with leading zeros.
34082 The default precision is 1. The result of converting zero with an explicit precision of
34083 zero shall be no characters.

34084 o The **unsigned** argument shall be converted to unsigned octal format in the style
34085 "*dddd*". The precision specifies the minimum number of digits to appear; if the value
34086 being converted can be represented in fewer digits, it shall be expanded with leading
34087 zeros. The default precision is 1. The result of converting zero with an explicit precision
34088 of zero shall be no characters.

34089 u The **unsigned** argument shall be converted to unsigned decimal format in the style
34090 "*dddd*". The precision specifies the minimum number of digits to appear; if the value
34091 being converted can be represented in fewer digits, it shall be expanded with leading
34092 zeros. The default precision is 1. The result of converting zero with an explicit precision
34093 of zero shall be no characters.

34094 x The **unsigned** argument shall be converted to unsigned hexadecimal format in the style
34095 "*dddd*"; the letters "abcdef" are used. The precision specifies the minimum number
34096 of digits to appear; if the value being converted can be represented in fewer digits, it
34097 shall be expanded with leading zeros. The default precision is 1. The result of
34098 converting zero with an explicit precision of zero shall be no characters.

34099 X Equivalent to the x conversion specifier, except that letters "ABCDEF" are used instead
34100 of "abcdef".

34101 f, F The **double** argument shall be converted to decimal notation in the style
34102 "[−]*ddd.ddd*", where the number of digits after the radix character is equal to the
34103 precision specification. If the precision is missing, it shall be taken as 6; if the precision
34104 is explicitly zero and no '#' flag is present, no radix character shall appear. If a radix
34105 character appears, at least one digit appears before it. The low-order digit shall be
34106 rounded in an implementation-defined manner.

34107 A **double** argument representing an infinity shall be converted in one of the styles
34108 "[−]inf" or "[−]infinity"; which style is implementation-defined. A **double**
34109 argument representing a NaN shall be converted in one of the styles "[−]nan(*n−*
34110 *char−sequence*)" or "[−]nan"; which style, and the meaning of any *n-char-*
34111 *sequence*, is implementation-defined. The F conversion specifier produces "INF",
34112 "INFINITY", or "NAN" instead of "inf", "infinity", or "nan", respectively.

34113 e, E The **double** argument shall be converted in the style "[−]*d.ddd*e±*dd*", where there is
34114 one digit before the radix character (which is non-zero if the argument is non-zero) and
34115 the number of digits after it is equal to the precision; if the precision is missing, it shall
34116 be taken as 6; if the precision is zero and no '#' flag is present, no radix character shall
34117 appear. The low-order digit shall be rounded in an implementation-defined manner.
34118 The E conversion specifier shall produce a number with 'E' instead of 'e'
34119 introducing the exponent. The exponent shall always contain at least two digits. If the
34120 value is zero, the exponent shall be zero.

34121 A **double** argument representing an infinity or NaN shall be converted in the style of
34122 an f or F conversion specifier.

34123 g, G The **double** argument representing a floating-point number shall be converted in the
34124 style f or e (or in the style F or E in the case of a G conversion specifier), depending on
34125 the value converted and the precision. Let P equal the precision if non-zero, 6 if the
34126 precision is omitted, or 1 if the precision is zero. Then, if a conversion with style E
34127 would have an exponent of *X*:

34128 — If P>*X*≥−4, the conversion shall be with style f (or F) and precision P−(*X*+1).

34129 — Otherwise, the conversion shall be with style e (or E) and precision P−1.

34130 Finally, unless the '#' flag is used, any trailing zeros shall be removed from the
34131 fractional portion of the result and the decimal-point character shall be removed if there
34132 is no fractional portion remaining.

34133 A **double** argument representing an infinity or NaN shall be converted in the style of
34134 an f or F conversion specifier.

34135 a, A A **double** argument representing a floating-point number shall be converted in the
34136 style "[−]0x*h.hhhh*p±*d*", where there is one hexadecimal digit (which shall be non-
34137 zero if the argument is a normalized floating-point number and is otherwise
34138 unspecified) before the decimal-point character and the number of hexadecimal digits
34139 after it is equal to the precision; if the precision is missing and FLT_RADIX is a power
34140 of 2, then the precision shall be sufficient for an exact representation of the value; if the
34141 precision is missing and FLT_RADIX is not a power of 2, then the precision shall be
34142 sufficient to distinguish values of type **double**, except that trailing zeros may be
34143 omitted; if the precision is zero and the '#' flag is not specified, no decimal-point
34144 character shall appear. The letters "abcdef" shall be used for a conversion and the
34145 letters "ABCDEF" for A conversion. The A conversion specifier produces a number with
34146 'X' and 'P' instead of 'x' and 'p'. The exponent shall always contain at least one
34147 digit, and only as many more digits as necessary to represent the decimal exponent of

2. If the value is zero, the exponent shall be zero.

A **double** argument representing an infinity or NaN shall be converted in the style of an `f` or `F` conversion specifier.

`c` The **int** argument shall be converted to an **unsigned char**, and the resulting byte shall be written.

CX If an `l` (ell) qualifier is present, the **wint_t** argument shall be converted to a multi-byte sequence as if by a call to *wcrtomb*( ) with a pointer to storage of at least MB_CUR_MAX bytes, the **wint_t** argument converted to **wchar_t**, and an initial shift state, and the resulting byte(s) written.

`s` The argument shall be a pointer to an array of **char**. Bytes from the array shall be written up to (but not including) any terminating null byte. If the precision is specified, no more than that many bytes shall be written. If the precision is not specified or is greater than the size of the array, the application shall ensure that the array contains a null byte.

If an `l` (ell) qualifier is present, the argument shall be a pointer to an array of type **wchar_t**. Wide characters from the array shall be converted to characters (each as if by a call to the *wcrtomb*( ) function, with the conversion state described by an **mbstate_t** object initialized to zero before the first wide character is converted) up to and including a terminating null wide character. The resulting characters shall be written up to (but not including) the terminating null character (byte). If no precision is specified, the application shall ensure that the array contains a null wide character. If a precision is specified, no more than that many characters (bytes) shall be written (including shift sequences, if any), and the array shall contain a null wide character if, to equal the character sequence length given by the precision, the function would need to access a wide character one past the end of the array. In no case shall a partial character be written.

`p` The argument shall be a pointer to **void**. The value of the pointer is converted to a sequence of printable characters, in an implementation-defined manner.

`n` The argument shall be a pointer to an integer into which is written the number of bytes written to the output so far by this call to one of the *fprintf*( ) functions. No argument is converted.

XSI `C` Equivalent to `lc`.

XSI `S` Equivalent to `ls`.

`%` Write a `'%'` character; no argument shall be converted. The application shall ensure that the complete conversion specification is `%%`.

If a conversion specification does not match one of the above forms, the behavior is undefined. If any argument is not the correct type for the corresponding conversion specification, the behavior is undefined.

In no case shall a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field shall be expanded to contain the conversion result. Characters generated by *fprintf*( ) and *printf*( ) are printed as if *fputc*( ) had been called.

For the `a` and `A` conversion specifiers, if FLT_RADIX is a power of 2, the value shall be correctly rounded to a hexadecimal floating number with the given precision.

For `a` and `A` conversions, if FLT_RADIX is not a power of 2 and the result is not exactly representable in the given precision, the result should be one of the two adjacent numbers in

hexadecimal floating style with the given precision, with the extra stipulation that the error should have a correct sign for the current rounding direction.

For the `e`, `E`, `f`, `F`, `g`, and `G` conversion specifiers, if the number of significant decimal digits is at most DECIMAL_DIG, then the result should be correctly rounded. If the number of significant decimal digits is more than DECIMAL_DIG but the source value is exactly representable with DECIMAL_DIG digits, then the result should be an exact representation with trailing zeros. Otherwise, the source value is bounded by two adjacent decimal strings $L < U$, both having DECIMAL_DIG significant digits; the value of the resultant decimal string D should satisfy $L <= D <= U$, with the extra stipulation that the error should have a correct sign for the current rounding direction.

CX The last data modification and last file status change timestamps of the file shall be marked for update:

1. Between the call to a successful execution of *fprintf*( ) or *printf*( ) and the next successful completion of a call to *fflush*( ) or *fclose*( ) on the same stream or a call to *exit*( ) or *abort*( )
2. Upon successful completion of a call to *dprintf*( )

RETURN VALUE

CX Upon successful completion, the *dprintf*( ), *fprintf*( ), and *printf*( ) functions shall return the number of bytes transmitted.

CX Upon successful completion, the *asprintf*( ) function shall return the number of bytes written to the allocated string stored in the location referenced by *ptr*, excluding the terminating null byte.

Upon successful completion, the *sprintf*( ) function shall return the number of bytes written to *s*, excluding the terminating null byte.

Upon successful completion, the *snprintf*( ) function shall return the number of bytes that would be written to *s* had *n* been sufficiently large excluding the terminating null byte.

CX If an error was encountered, these functions shall return a negative value and set *errno* to indicate the error. For *asprintf*( ), if memory allocation was not possible, or if some other error occurs, the function shall return a negative value, and the contents of the location referenced by *ptr* are undefined, but shall not refer to allocated memory.

If the value of *n* is zero on a call to *snprintf*( ), nothing shall be written, the number of bytes that would have been written had *n* been sufficiently large excluding the terminating null shall be returned, and *s* may be a null pointer.

ERRORS

CX For the conditions under which *dprintf*( ), *fprintf*( ), and *printf*( ) fail and may fail, refer to *fputc*( ) or *fputwc*( ).

In addition, all forms of *fprintf*( ) shall fail if:

CX [EILSEQ] A wide-character code that does not correspond to a valid character has been detected.

CX [EOVERFLOW] The value to be returned is greater than {INT_MAX}.

CX The *asprintf*( ) function shall fail if:

[ENOMEM] Insufficient storage space is available.

The *dprintf*( ) function may fail if:

34234 [EBADF] The *fildes* argument is not a valid file descriptor.

34235 CX The *dprintf*( ), *fprintf*( ), and *printf*( ) functions may fail if:

34236 CX [ENOMEM] Insufficient storage space is available.

34237 **EXAMPLES**

34238 **Printing Language-Independent Date and Time**

34239 The following statement can be used to print date and time using a language-independent
34240 format:

34241 ```
printf(format, weekday, month, day, hour, min);
```

34242 For American usage, *format* could be a pointer to the following string:

34243 ```
"%s, %s %d, %d:%.2d\n"
```

34244 This example would produce the following message:

34245 ```
Sunday, July 3, 10:02
```

34246 For German usage, *format* could be a pointer to the following string:

34247 ```
"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

34248 This definition of *format* would produce the following message:

34249 ```
Sonntag, 3. Juli, 10:02
```

34250 **Printing File Information**

34251 The following example prints information about the type, permissions, and number of links of a
34252 specific file in a directory.

34253 The first two calls to *printf*( ) use data decoded from a previous *stat*( ) call. The user-defined
34254 *strperm*( ) function shall return a string similar to the one at the beginning of the output for the
34255 following command:

34256 ```
ls -l
```

34257 The next call to *printf*( ) outputs the owner's name if it is found using *getpwuid*( ); the *getpwuid*( )
34258 function shall return a **passwd** structure from which the name of the user is extracted. If the user
34259 name is not found, the program instead prints out the numeric value of the user ID.

34260 The next call prints out the group name if it is found using *getgrgid*( ); *getgrgid*( ) is very similar
34261 to *getpwuid*( ) except that it shall return group information based on the group number. Once
34262 again, if the group is not found, the program prints the numeric value of the group for the entry.

34263 The final call to *printf*( ) prints the size of the file.

```
34264 #include <stdio.h>
34265 #include <sys/types.h>
34266 #include <pwd.h>
34267 #include <grp.h>

34268 char *strperm (mode_t);
34269 ...
34270 struct stat statbuf;
34271 struct passwd *pwd;
```



```
34272         struct group *grp;
34273         ...
34274         printf("%10.10s", strperm (statbuf.st_mode));
34275         printf("%4d", statbuf.st_nlink);
34276         if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
34277             printf(" %-8.8s", pwd->pw_name);
34278         else
34279             printf(" %-8ld", (long) statbuf.st_uid);
34280         if ((grp = getgrgid(statbuf.st_gid)) != NULL)
34281             printf(" %-8.8s", grp->gr_name);
34282         else
34283             printf(" %-8ld", (long) statbuf.st_gid);
34284         printf("%9jd", (intmax_t) statbuf.st_size);
34285         ...
```

34286 **Printing a Localized Date String**

34287 The following example gets a localized date string. The *nl_langinfo*( ) function shall return the
34288 localized date string, which specifies the order and layout of the date. The *strftime*( ) function
34289 takes this information and, using the **tm** structure for values, places the date and time
34290 information into *datestring*. The *printf*( ) function then outputs *datestring* and the name of the
34291 entry.

```
34292         #include <stdio.h>
34293         #include <time.h>
34294         #include <langinfo.h>
34295         ...
34296         struct dirent *dp;
34297         struct tm *tm;
34298         char datestring[256];
34299         ...
34300         strftime(datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
34301         printf(" %s %s\n", datestring, dp->d_name);
34302         ...
```

34303 **Printing Error Information**

34304 The following example uses *fprintf*( ) to write error information to standard error.

34305 In the first group of calls, the program tries to open the password lock file named **LOCKFILE**. If
34306 the file already exists, this is an error, as indicated by the O_EXCL flag on the *open*( ) function. If
34307 the call fails, the program assumes that someone else is updating the password file, and the
34308 program exits.

34309 The next group of calls saves a new password file as the current password file by creating a link
34310 between **LOCKFILE** and the new password file **PASSWDFILE**.

```
34311         #include <sys/types.h>
34312         #include <sys/stat.h>
34313         #include <fcntl.h>
34314         #include <stdio.h>
34315         #include <stdlib.h>
```



```
#include <unistd.h>
#include <string.h>
#include <errno.h>

#define LOCKFILE "/etc/ptmp"
#define PASSWDFILE "/etc/passwd"
...
int pfd;
...
if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
{
    fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
    exit(1);
}
...
if (link(LOCKFILE,PASSWDFILE) == -1) {
    fprintf(stderr, "Link error: %s\n", strerror(errno));
    exit(1);
}
...
```

Printing Usage Information

The following example checks to make sure the program has the necessary arguments, and uses *fprintf*( ) to print usage information if the expected number of arguments is not present.

```
#include <stdio.h>
#include <stdlib.h>
...
char *Options = "hdbtl";
...
if (argc < 2) {
    fprintf(stderr, "Usage: %s -%s <file\n", argv[0], Options); exit(1);
}
...
```

Formatting a Decimal String

The following example prints a key and data pair on *stdout*. Note use of the <asterisk> (`'*'`) in the format string; this ensures the correct number of decimal places for the element based on the number of elements requested.

```
#include <stdio.h>
...
long i;
char *keystr;
int elementlen, len;
...
while (len < elementlen) {
...
    printf("%s Element%0*ld\n", keystr, elementlen, i);
...
```



```
}
```

Creating a Pathname

The following example creates a pathname using information from a previous *getpwnam*( ) function that returned the password database entry of the user.

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
...
char *pathname;
struct passwd *pw;
size_t len;
...
// digits required for pid_t is number of bits times
// log2(10) = approx 10/33
len = strlen(pw->pw_dir) + 1 + 1+(sizeof(pid_t)*80+32)/33 +
    sizeof ".out";
pathname = malloc(len);
if (pathname != NULL)
{
    snprintf(pathname, len, "%s/%jd.out", pw->pw_dir,
        (intmax_t)getpid());
    ...
}
```

Reporting an Event

The following example loops until an event has timed out. The *pause*( ) function waits forever unless it receives a signal. The *fprintf*( ) statement should never occur due to the possible return values of *pause*( ).

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
...
while (!event_complete) {
...
    if (pause() != -1 || errno != EINTR)
        fprintf(stderr, "pause: unknown error: %s\n", strerror(errno));
}
...
```


Printing Monetary Information

The following example uses *strfmon*( ) to convert a number and store it as a formatted monetary string named *convbuf*. If the first number is printed, the program prints the format and the description; otherwise, it just prints the number.

```
#include <monetary.h>
#include <stdio.h>
...
struct tblfmt {
    char *format;
    char *description;
};

struct tblfmt table[] = {
    { "%n", "default formatting" },
    { "%11n", "right align within an 11 character field" },
    { "%#5n", "aligned columns for values up to 99999" },
    { "%=*#5n", "specify a fill character" },
    { "%=0#5n", "fill characters do not use grouping" },
    { "%^#5n", "disable the grouping separator" },
    { "%^#5.0n", "round off to whole units" },
    { "%^#5.4n", "increase the precision" },
    { "%(#5n", "use an alternative pos/neg style" },
    { "%!(#5n", "disable the currency symbol" },
};
...
float input[3];
int i, j;
char convbuf[100];
...
strfmon(convbuf, sizeof(convbuf), table[i].format, input[j]);

if (j == 0) {
    printf("%s%s%s\n", table[i].format,
        convbuf, table[i].description);
}
else {
    printf("%s\n", convbuf);
}
...
```

Printing Wide Characters

The following example prints a series of wide characters. Suppose that `"L`@`"` expands to three bytes:

```
wchar_t wz [3] = L"@@";       // Zero-terminated
wchar_t wn [3] = L"@@@";      // Unterminated

fprintf (stdout,"%ls", wz);   // Outputs 6 bytes
fprintf (stdout,"%ls", wn);   // Undefined because wn has no terminator
fprintf (stdout,"%4ls", wz);  // Outputs 3 bytes
fprintf (stdout,"%4ls", wn);  // Outputs 3 bytes; no terminator needed
fprintf (stdout,"%9ls", wz);  // Outputs 6 bytes
```



```
fprintf (stdout,"%9ls", wn);  // Outputs 9 bytes; no terminator needed
fprintf (stdout,"%10ls", wz); // Outputs 6 bytes
fprintf (stdout,"%10ls", wn); // Undefined because wn has no terminator
```

In the last line of the example, after processing three characters, nine bytes have been output. The fourth character must then be examined to determine whether it converts to one byte or more. If it converts to more than one byte, the output is only nine bytes. Since there is no fourth character in the array, the behavior is undefined.

APPLICATION USAGE

If the application calling *fprintf*( ) has any objects of type **wint_t** or **wchar_t**, it must also include the **<wchar.h>** header to have these objects defined.

The space allocated by a successful call to *asprintf*( ) should be subsequently freed by a call to *free*( ).

RATIONALE

If an implementation detects that there are insufficient arguments for the format, it is recommended that the function should fail and report an [EINVAL] error.

The behavior specified for the `%lc` conversion differs slightly from the specification in the ISO C standard, in that printing the null wide character produces a null byte instead of 0 bytes of output as would be required by a strict reading of the ISO C standard's direction to behave as if applying the `%ls` specifier to a **wchar_t** array whose first element is the null wide character. Requiring a multi-byte output for every possible wide character, including the null character, matches historical practice, and provides consistency with `%c` in *fprintf*( ) and with both `%c` and `%lc` in *fwprintf*( ). It is anticipated that a future edition of the ISO C standard will change to match the behavior specified here.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fputc*( ), *fscanf*( ), *setlocale*( ), *strfmon*( ), *strlcat*( ), *wcrtomb*( ), *wcslcat*( )

XBD Chapter 7 (on page 127), **<inttypes.h>**, **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the `l` (ell) qualifier can now be used with `c` and `s` conversion specifiers.

The *snprintf*( ) function is new in Issue 5.

Issue 6

Extensions beyond the ISO C standard are marked.

The normative text is updated to avoid use of the term ``must'' for application requirements.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The prototypes for *fprintf*( ), *printf*( ), *snprintf*( ), and *sprintf*( ) are updated, and the XSI shading is removed from *snprintf*( ).
- The description of *snprintf*( ) is aligned with the ISO C standard. Note that this supersedes the *snprintf*( ) description in The Open Group Base Resolution bwg98-006, which changed the behavior from Issue 5.

- The DESCRIPTION is updated.

The DESCRIPTION is updated to use the terms ``conversion specifier'' and ``conversion specification'' consistently.

ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

An example of printing wide characters is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #161 is applied, updating the DESCRIPTION of the 0 flag.

Austin Group Interpretation 1003.1-2001 #170 is applied.

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #68 (SD5-XSH-ERN-70) is applied, revising the description of `g` and `G`.

SD5-XSH-ERN-174 is applied.

The *dprintf*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Functionality relating to the `%n$` form of conversion specification and the **<apostrophe>** flag is moved from the XSI option to the Base.

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0163 [302], XSH/TC1-2008/0164 [316], XSH/TC1-2008/0165 [316], XSH/TC1-2008/0166 [451,291], and XSH/TC1-2008/0167 [14] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0126 [894], XSH/TC2-2008/0127 [557], and XSH/TC2-2008/0128 [936] are applied.

Issue 8

Austin Group Defect 986 is applied, adding *strlcat*( ) and *wcslcat*( ) to the SEE ALSO section.

Austin Group Defect 1020 is applied, clarifying that the *snprintf*( ) argument *n* limits the number of bytes written to *s*; it is not necessarily the same as the size of *s*.

Austin Group Defect 1021 is applied, changing ``output error'' to ``error'' in the RETURN VALUE section.

Austin Group Defect 1137 is applied, clarifying the use of "`%`*n*`$`" and "`*`*m*`$`" in conversion specifications.

Austin Group Defect 1205 is applied, changing the description of the `%` conversion specifier.

Austin Group Defect 1219 is applied, removing the *snprintf*( )-specific [EOVERFLOW] error.

Austin Group Defect 1496 is applied, adding the *asprintf*( ) function.

Austin Group Defect 1562 is applied, clarifying that it is the application's responsibility to ensure that the format is a character string, beginning and ending in its initial shift state, if any.

Austin Group Defect 1647 is applied, changing the description of the `c` conversion specifier and updating the statement that this volume of POSIX.1-2024 defers to the ISO C standard so that it excludes the `%lc` conversion when passed a null wide character.

34531 **NAME**

34532 fputc — put a byte on a stream

34533 **SYNOPSIS**

34534 ```
#include <stdio.h>
```

34535 ```
int fputc(int c, FILE *stream);
```

34536 **DESCRIPTION**

34537 CX The functionality described on this reference page is aligned with the ISO C standard. Any
34538 conflict between the requirements described here and the ISO C standard is unintentional. This
34539 volume of POSIX.1-2024 defers to the ISO C standard.

34540 The *fputc*( ) function shall write the byte specified by *c* (converted to an **unsigned char**) to the
34541 output stream pointed to by *stream*, at the position indicated by the associated file-position
34542 indicator for the stream (if defined), and shall advance the indicator appropriately. If the file
34543 cannot support positioning requests, or if the stream was opened with append mode, the byte
34544 shall be appended to the output stream.

34545 CX The last data modification and last file status change timestamps of the file shall be marked for
34546 update between the successful execution of *fputc*( ) and the next successful completion of a call
34547 to *fflush*( ) or *fclose*( ) on the same stream or a call to *exit*( ) or *abort*( ).

34548 **RETURN VALUE**

34549 Upon successful completion, *fputc*( ) shall return the value it has written. Otherwise, it shall
34550 CX return EOF, the error indicator for the stream shall be set, and *errno* shall be set to indicate the
34551 error.

34552 **ERRORS**

34553 The *fputc*( ) function shall fail if either the *stream* is unbuffered or the *stream*'s buffer needs to be
34554 flushed, and:

34555 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and
34556 the thread would be delayed in the write operation.

34557 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
34558 writing.

34559 CX [EFBIG] An attempt was made to write to a file that exceeds the maximum file size.

34560 CX [EFBIG] An attempt was made to write to a file that exceeds the file size limit of the
34561 process.
34562 XSI A SIGXFSZ signal shall also be generated for the thread.

34563 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
34564 offset maximum.

34565 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
34566 was transferred.

34567 CX [EIO] A physical I/O error has occurred, or the process is a member of a background
34568 process group attempting to write to its controlling terminal, TOSTOP is set,
34569 the calling thread is not blocking SIGTTOU, the process is not ignoring
34570 SIGTTOU, and the process group of the process is orphaned. This error may
34571 also be returned under implementation-defined conditions.

34572 CX [ENOSPC] There was no free space remaining on the device containing the file.

34573 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
34574 any process. A SIGPIPE signal shall also be sent to the thread.

34575 The *fputc*( ) function may fail if:

34576 CX [ENOMEM] Insufficient storage space is available.

34577 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
34578 capabilities of the device.

34579 **EXAMPLES**

34580 None.

34581 **APPLICATION USAGE**

34582 None.

34583 **RATIONALE**

34584 None.

34585 **FUTURE DIRECTIONS**

34586 None.

34587 **SEE ALSO**

34588 Section 2.5 (on page 521), *ferror*( ), *fopen*( ), *getrlimit*( ), *putc*( ), *puts*( ), *setbuf*( )

34589 XBD **<stdio.h>**

34590 **CHANGE HISTORY**

34591 First released in Issue 1. Derived from Issue 1 of the SVID.

34592 **Issue 5**

34593 Large File Summit extensions are added.

34594 **Issue 6**

34595 Extensions beyond the ISO C standard are marked.

34596 The following new requirements on POSIX implementations derive from alignment with the
34597 Single UNIX Specification:

34598 • The [EIO] and [EFBIG] mandatory error conditions are added.

34599 • The [ENOMEM] and [ENXIO] optional error conditions are added.

34600 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/37 is applied, updating the [EAGAIN]
34601 error in the ERRORS section from ``the process would be delayed'' to ``the thread would be
34602 delayed''.

34603 **Issue 7**

34604 Changes are made related to support for finegrained timestamps.

34605 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0168 [79] and XSH/TC1-2008/0169
34606 [14] are applied.

34607 **Issue 8**

34608 Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

34609 Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error
34610 relating to the file size limit for the process.

NAME

fputs — put a string on a stream

SYNOPSIS

```
#include <stdio.h>

int fputs(const char *restrict s, FILE *restrict stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fputs*( ) function shall write the null-terminated string pointed to by *s* to the stream pointed to by *stream*. The terminating null byte shall not be written.

CX The last data modification and last file status change timestamps of the file shall be marked for update between the successful execution of *fputs*( ) and the next successful completion of a call to *fflush*( ) or *fclose*( ) on the same stream or a call to *exit*( ) or *abort*( ).

RETURN VALUE

Upon successful completion, *fputs*( ) shall return a non-negative number. Otherwise, it shall return EOF, CX set an error indicator for the stream, and set *errno* to indicate the error.

ERRORS

Refer to *fputc*( ).

EXAMPLES

Printing to Standard Output

The following example gets the current time, converts it to a string using *localtime*( ) and *asctime*( ), and prints it to standard output using *fputs*( ). It then prints the number of minutes to an event for which it is waiting.

```
#include <time.h>
#include <stdio.h>
...
time_t now;
int minutes_to_event;
...
time(&now);
printf("The time is ");
fputs(asctime(localtime(&now)), stdout);
printf("There are still %d minutes to the event.\n",
    minutes_to_event);
...
```

APPLICATION USAGE

The *puts*( ) function appends a <newline> while *fputs*( ) does not.

This volume of POSIX.1-2024 requires that successful completion simply return a non-negative integer. There are at least three known different implementation conventions for this requirement:

- Return a constant value.

- Return the last character written.
- Return the number of bytes written. Note that this implementation convention cannot be adhered to for strings longer than {INT_MAX} bytes as the value would not be representable in the return type of the function. For backwards-compatibility, implementations can return the number of bytes for strings of up to {INT_MAX} bytes, and return {INT_MAX} for all longer strings.

RATIONALE

The *fputs*( ) function is one whose source code was specified in the referenced *The C Programming Language*. In the original edition, the function had no defined return value, yet many practical implementations would, as a side-effect, return the value of the last character written as that was the value remaining in the accumulator used as a return value. In the second edition of the book, either the fixed value 0 or EOF would be returned depending upon the return value of *ferror*( ); however, for compatibility with extant implementations, several implementations would, upon success, return a positive value representing the last byte written.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fopen*( ), *putc*( ), *puts*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

The *fputs*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0170 [174,412], XSH/TC1-2008/0171 [412], and XSH/TC1-2008/0172 [14] are applied.

NAME

fputwc — put a wide-character code on a stream

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

wint_t fputwc(wchar_t wc, FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fputwc*( ) function shall write the character corresponding to the wide-character code *wc* to the output stream pointed to by *stream*, at the position indicated by the associated file-position indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot support positioning requests, or if the stream was opened with append mode, the character is appended to the output stream. If an error occurs while writing the character, the shift state of the output file is left in an undefined state.

CX The last data modification and last file status change timestamps of the file shall be marked for update between the successful execution of *fputwc*( ) and the next successful completion of a call to *fflush*( ) or *fclose*( ) on the same stream or a call to *exit*( ) or *abort*( ).

The *fputwc*( ) function shall not change the setting of *errno* if successful.

RETURN VALUE

CX Upon successful completion, *fputwc*( ) shall return *wc*. Otherwise, it shall return WEOF, *errno* shall be set to indicate the error, and for errors other than [EILSEQ] the error indicator for the stream shall be set; CX the error indicator for the stream shall also be set for [EILSEQ] errors.

ERRORS

The *fputwc*( ) function shall fail if either the stream is unbuffered or data in the *stream*'s buffer needs to be written, and:

CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the thread would be delayed in the write operation.

CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for writing.

CX [EFBIG] An attempt was made to write to a file that exceeds the maximum file size.

CX [EFBIG] An attempt was made to write to a file that exceeds the file size limit of the process.
XSI A SIGXFSZ signal shall also be generated for the thread.

CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.

[EILSEQ] The wide-character code *wc* does not correspond to a valid character.

CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data was transferred.

CX [EIO] A physical I/O error has occurred, or the process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the calling thread is not blocking SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the process is orphaned. This error may

also be returned under implementation-defined conditions.

CX [ENOSPC] There was no free space remaining on the device containing the file.

CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal shall also be sent to the thread.

The *fputwc*( ) function may fail if:

CX [ENOMEM] Insufficient storage space is available.

CX [ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The requirement to set the error indicator for the stream on [EILSEQ] errors is CX shaded because the ISO C standard does not require it to be set for *fputwc*( ) encoding errors, although it does for *fgetwc*( ). The next revision of the ISO C standard is expected to address this inconsistency by requiring the error indicator for the stream to be set for *fputwc*( ) encoding errors.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *ferror*( ), *fopen*( ), *getrlimit*( ), *setbuf*( )

XBD **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc* is changed from **wint_t** to **wchar_t**.

The Optional Header (OH) marking is removed from **<stdio.h>**.

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EFBIG] and [EIO] mandatory error conditions are added.
- The [ENOMEM] and [ENXIO] optional error conditions are added.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/38 is applied, updating the [EAGAIN] error in the ERRORS section from ``the process would be delayed'' to ``the thread would be delayed''.

34764 **Issue 7**

34765 Changes are made related to support for finegrained timestamps.

34766 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0173 [105], XSH/TC1-2008/0174 [79],
34767 and XSH/TC1-2008/0175 [14] are applied.

34768 **Issue 8**

34769 Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

34770 Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error
34771 relating to the file size limit for the process.

34772 Austin Group Defect 1769 is applied, changing the CX shading in the RETURN VALUE section.

NAME

fputws — put a wide-character string on a stream

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int fputws(const wchar_t *restrict ws, FILE *restrict stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fputws*( ) function shall write a character string corresponding to the (null-terminated) wide-character string pointed to by *ws* to the stream pointed to by *stream*. No character corresponding to the terminating null wide-character code shall be written.

CX The last data modification and last file status change timestamps of the file shall be marked for update between the successful execution of *fputws*( ) and the next successful completion of a call to *fflush*( ) or *fclose*( ) on the same stream or a call to *exit*( ) or *abort*( ).

RETURN VALUE

Upon successful completion, *fputws*( ) shall return a non-negative number. Otherwise, it shall return –1, set an error indicator for the stream, CX and set *errno* to indicate the error.

ERRORS

Refer to *fputwc*( ).

EXAMPLES

None.

APPLICATION USAGE

The *fputws*( ) function does not append a <newline>.

This volume of POSIX.1-2024 requires that successful completion simply return a non-negative integer. There are at least three known different implementation conventions for this requirement:

- Return a constant value.
- Return the last character written.
- Return the number of bytes written. Note that this implementation convention cannot be adhered to for strings longer than {INT_MAX} bytes as the value would not be representable in the return type of the function. For backwards-compatibility, implementations can return the number of bytes for strings of up to {INT_MAX} bytes, and return {INT_MAX} for all longer strings.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fopen*( )

XBD **<stdio.h>**, **<wchar.h>**

34815 **CHANGE HISTORY**

34816 First released in Issue 4. Derived from the MSE working draft.

34817 **Issue 5**

34818 The Optional Header (OH) marking is removed from **<stdio.h>**.

34819 **Issue 6**

34820 Extensions beyond the ISO C standard are marked.

34821 The *fputws*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

34822 **Issue 7**

34823 Changes are made related to support for finegrained timestamps.

34824 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0176 [412] and XSH/TC1-2008/0177
34825 [14] are applied.

NAME

fread — binary input

SYNOPSIS

```
#include <stdio.h>

size_t fread(void *restrict ptr, size_t size, size_t nitems,
    FILE *restrict stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fread*( ) function shall read into the array pointed to by *ptr* up to *nitems* elements whose size is specified by *size* in bytes, from the stream pointed to by *stream*. For each object, *size* calls shall be made to the *fgetc*( ) function and the results stored, in the order read, in an array of **unsigned char** exactly overlaying the object. The file position indicator for the stream (if defined) shall be advanced by the number of bytes successfully read. If an error occurs, the resulting value of the file position indicator for the stream is unspecified. If a partial element is read, its value is unspecified.

CX The *fread*( ) function may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetc*( ), *fgets*( ), *fread*( ), *fscanf*( ), *getc*( ), *getchar*( ), *getdelim*( ), *getline*( ), or *scanf*( ) using *stream* that returns data not supplied by a prior call to *ungetc*( ).

RETURN VALUE

The *fread*( ) function shall return the number of elements successfully read, which shall be less than *nitems* only if an error or end-of-file is encountered, or *size* is zero. If *size* or *nitems* is 0, *fread*( ) shall return 0 and the contents of the array and the state of the stream shall remain unchanged. Otherwise, if an error occurs, the error indicator for the stream shall be set, and *errno* shall be set to indicate the error.

CX

ERRORS

Refer to *fgetc*( ).

EXAMPLES

Reading from a Stream

The following example transfers a single 100-byte fixed length record from the *fp* stream into the array pointed to by *buf*.

```
#include <stdio.h>
...
size_t elements_read;
char buf[100];
FILE *fp;
...
elements_read = fread(buf, sizeof(buf), 1, fp);
...
```

If a read error occurs, *elements_read* will be zero but the number of bytes read from the stream could be anything from zero to *sizeof*(*buf*)–1.

The following example reads multiple single-byte elements from the *fp* stream into the array pointed to by *buf*.


```
#include <stdio.h>
...
size_t bytes_read;
char buf[100];
FILE *fp;
...
bytes_read = fread(buf, 1, sizeof(buf), fp);
...
```

If a read error occurs, *bytes_read* will contain the number of bytes read from the stream.

APPLICATION USAGE

The *ferror*( ) or *feof*( ) functions must be used to distinguish between an error condition and an end-of-file condition.

Because of possible differences in element length and byte ordering, files written using *fwrite*( ) are application-dependent, and possibly cannot be read using *fread*( ) by a different application or by the same application on a different processor.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *feof*( ), *ferror*( ), *fgetc*( ), *fopen*( ), *fscanf*( ), *getc*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *fread*( ) prototype is updated.
- The DESCRIPTION is updated to describe how the bytes from a call to *fgetc*( ) are stored.

Issue 7

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0178 [232] and XSH/TC1-2008/0179 [14] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0129 [926] is applied.

Issue 8

Austin Group Defect 1196 is applied, clarifying the RETURN VALUE section.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1624 is applied, changing the RETURN VALUE section.

34909 **NAME**

34910 free — free allocated memory

34911 **SYNOPSIS**

34912 `#include <stdlib.h>`

34913 `void free(void *ptr);`

34914 **DESCRIPTION**

34915 CX The functionality described on this reference page is aligned with the ISO C standard. Any
34916 conflict between the requirements described here and the ISO C standard is unintentional. This
34917 volume of POSIX.1-2024 defers to the ISO C standard.

34918 The *free*( ) function shall cause the space pointed to by *ptr* to be deallocated; that is, made
34919 available for further allocation. If *ptr* is a null pointer, no action shall occur. Otherwise, if the
34920 argument does not match a pointer earlier returned by *aligned_alloc*( ), *calloc*( ), *malloc*( ),
34921 ADV *posix_memalign*( ), *realloc*( ),
34922 CX *reallocarray*( ), or a function in POSIX.1-2024 that allocates memory as if by *malloc*( ), or if the
34923 CX space has been deallocated by a call to *free*( ), *reallocarray*( ), or *realloc*( ), the behavior is
34924 undefined.

34925 Any use of a pointer that refers to freed space results in undefined behavior.

34926 CX The *free*( ) function shall not modify *errno* if *ptr* is a null pointer or a pointer previously returned
34927 as if by *malloc*( ) and not yet deallocated.

34928 For purposes of determining the existence of a data race, *free*( ) shall behave as though it
34929 accessed only memory locations accessible through its argument and not other static duration
34930 storage. The function may, however, visibly modify the storage that it deallocates. Calls to
34931 ADV *aligned_alloc*( ), *calloc*( ), *free*( ), *malloc*( ), *posix_memalign*( ),
34932 CX *reallocarray*( ), and *realloc*( ) that allocate or deallocate a particular region of memory shall occur
34933 in a single total order (see XBD Section 4.15.1, on page 100), and each such deallocation call shall
34934 synchronize with the next allocation (if any) in this order.

34935 **RETURN VALUE**

34936 The *free*( ) function shall not return a value.

34937 **ERRORS**

34938 No errors are defined.

34939 **EXAMPLES**

34940 None.

34941 **APPLICATION USAGE**

34942 There is now no requirement for the implementation to support the inclusion of **<malloc.h>**.

34943 Because the *free*( ) function does not modify *errno* for valid pointers, it is safe to use it in cleanup
34944 code without corrupting earlier errors, such as in this example code:

```
34945     // buf was obtained by malloc(buflen)
34946     ret = write(fd, buf, buflen);
34947     if (ret < 0) {
34948         free(buf);
34949         return ret;
34950     }
```

34951 However, earlier versions of this standard did not require this, and the same example had to be
34952 written as:

```
34953     // buf was obtained by malloc(buflen)
```



```
        ret = write(fd, buf, buflen);
        if (ret < 0) {
            int save = errno;
            free(buf);
            errno = save;
            return ret;
        }
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aligned_alloc( ), *calloc*( ), *malloc*( ), *posix_memalign*( ), *realloc*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Reference to the *valloc*( ) function is removed.

Issue 7

The DESCRIPTION is updated to clarify that if the pointer returned is not by a function that allocates memory as if by *malloc*( ), then the behavior is undefined.

Issue 8

Austin Group Defect 385 is applied, adding a requirement that *free*( ) does not modify *errno* when passed a pointer to an object than can be freed.

Austin Group Defect 1218 is applied, adding *reallocarray*( ).

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018 standard.

NAME

freeaddrinfo, getaddrinfo — get address information

SYNOPSIS

```
#include <sys/socket.h>
#include <netdb.h>

void freeaddrinfo(struct addrinfo *ai);
int getaddrinfo(const char *restrict nodename,
    const char *restrict servname,
    const struct addrinfo *restrict hints,
    struct addrinfo **restrict res);
```

DESCRIPTION

The *freeaddrinfo*( ) function shall free one or more **addrinfo** structures returned by *getaddrinfo*( ), along with any additional storage associated with those structures. If the *ai_next* field of the structure is not null, the entire list of structures shall be freed. The *freeaddrinfo*( ) function shall support the freeing of arbitrary sublists of an **addrinfo** list originally returned by *getaddrinfo*( ). The *freeaddrinfo*( ) function shall not modify *errno* if *ai* is a sublist previously returned by *getaddrinfo*( ) and not yet freed.

The *getaddrinfo*( ) function shall translate the name of a service location (for example, a host name) and/or a service name and shall return a set of socket addresses and associated information to be used in creating a socket with which to address the specified service.

Note: In many cases it is implemented by the Domain Name System, as documented in RFC 1034, RFC 1035, and RFC 3596.

The *freeaddrinfo*( ) and *getaddrinfo*( ) functions shall be thread-safe.

The *nodename* and *servname* arguments are either null pointers or pointers to null-terminated strings. One or both of these two arguments shall be supplied by the application as a non-null pointer.

The format of a valid name depends on the address family or families. If a specific family is not given and the name could be interpreted as valid within multiple supported families, the implementation shall attempt to resolve the name in all supported families and, in absence of errors, one or more results shall be returned.

If the *nodename* argument is not null, it can be a descriptive name or can be an address string. If IP6 the specified address family is AF_INET, AF_INET6, or AF_UNSPEC, valid descriptive names include host names. If the specified address family is AF_INET or AF_UNSPEC, address strings using Internet standard dot notation as specified in *inet_ntop*( ) are valid.

IP6 If the specified address family is AF_INET6 or AF_UNSPEC, standard IPv6 text forms described in *inet_ntop*( ) are valid.

If *nodename* is not null, the requested service location is named by *nodename*; otherwise, the requested service location is local to the caller.

If *servname* is null, the call shall return network-level addresses for the specified *nodename.* If *servname* is not null, it is a null-terminated character string identifying the requested service. This can be either a descriptive name or a numeric representation suitable for use with the IP6 address family or families. If the specified address family is AF_INET, AF_INET6, or AF_UNSPEC, the service can be specified as a string specifying a decimal port number.

If the *hints* argument is not null, it refers to a structure containing input values that directs the operation by providing options and by limiting the returned information to a specific socket type, address family, and/or protocol, as described below. The application shall ensure that each

of the *ai_addrlen*, *ai_addr*, *ai_canonname*, and *ai_next* members, as well as each of the non-standard additional members, if any, of this *hints* structure is initialized. If any of these members has a value other than the value that would result from default initialization, the behavior is implementation-defined. A value of AF_UNSPEC for *ai_family* means that the caller shall accept any address family. A value of zero for *ai_socktype* means that the caller shall accept any socket type. A value of zero for *ai_protocol* means that the caller shall accept any protocol. If *hints* is a null pointer, the behavior shall be as if it referred to a structure containing the value zero for the *ai_flags*, *ai_socktype*, and *ai_protocol* fields, and AF_UNSPEC for the *ai_family* field.

The *ai_flags* field to which the *hints* parameter points shall be set to zero or be the bitwise-inclusive OR of one or more of the values AI_PASSIVE, AI_CANONNAME, AI_NUMERICHOST, AI_NUMERICSERV, AI_V4MAPPED, AI_ALL, and AI_ADDRCONFIG.

If the AI_PASSIVE flag is specified, the returned address information shall be suitable for use in binding a socket for accepting incoming connections for the specified service. In this case, if the *nodename* argument is null, then the IP address portion of the socket address structure shall be set to INADDR_ANY for an IPv4 address or IN6ADDR_ANY_INIT for an IPv6 address. If the AI_PASSIVE flag is not specified, the returned address information shall be suitable for a call to *connect*( ) (for a connection-mode protocol) or for a call to *connect*( ), *sendto*( ), or *sendmsg*( ) (for a connectionless protocol). In this case, if the *nodename* argument is null, then the IP address portion of the socket address structure shall be set to the loopback address. The AI_PASSIVE flag shall be ignored if the *nodename* argument is not null.

If the AI_CANONNAME flag is specified and the *nodename* argument is not null, the function shall attempt to determine the canonical name corresponding to *nodename* (for example, if *nodename* is an alias or shorthand notation for a complete name).

Note: Since different implementations use different conceptual models, the terms ``canonical name'' and ``alias'' cannot be precisely defined for the general case. However, Domain Name System implementations are expected to interpret them as they are used in RFC 1034.

A numeric host address string is not a ``name'', and thus does not have a ``canonical name'' form; no address to host name translation is performed. See below for handling of the case where a canonical name cannot be obtained.

If the AI_NUMERICHOST flag is specified, then a non-null *nodename* string supplied shall be a numeric host address string. Otherwise, an [EAI_NONAME] error is returned. This flag shall prevent any type of name resolution service (for example, the DNS) from being invoked.

If the AI_NUMERICSERV flag is specified, then a non-null *servname* string supplied shall be a numeric port string. Otherwise, an [EAI_NONAME] error shall be returned. This flag shall prevent any type of name resolution service (for example, NIS+) from being invoked.

IP6 By default, with an *ai_family* of AF_INET6, *getaddrinfo*( ) shall return only IPv6 addresses. If the AI_V4MAPPED flag is specified along with an *ai_family* of AF_INET6, then *getaddrinfo*( ) shall return IPv4-mapped IPv6 addresses on finding no matching IPv6 addresses. The AI_V4MAPPED flag shall be ignored unless *ai_family* equals AF_INET6. If the AI_ALL flag is used with the AI_V4MAPPED flag, then *getaddrinfo*( ) shall return all matching IPv6 and IPv4 addresses. The AI_ALL flag without the AI_V4MAPPED flag shall be ignored.

If the AI_ADDRCONFIG flag is specified, IPv4 addresses shall be returned only if an IPv4 IP6 address is configured on the local system, and IPv6 addresses shall be returned only if an IPv6 address is configured on the local system.

The *ai_socktype* field to which argument *hints* points specifies the socket type for the service, as defined in Section 2.10.6 (on page 550). If a specific socket type is not given (for example, a value of zero) and the service name could be interpreted as valid with multiple supported socket

types, the implementation shall attempt to resolve the service name for all supported socket types and, in the absence of errors, all possible results shall be returned. A non-zero socket type value shall limit the returned information to values with the specified socket type.

If the *ai_family* field to which *hints* points has the value AF_UNSPEC, addresses shall be returned for use with any address family that can be used with the specified *nodename* and/or *servname*. Otherwise, addresses shall be returned for use only with the specified address family. If *ai_family* is not AF_UNSPEC and *ai_protocol* is not zero, then addresses shall be returned for use only with the specified address family and protocol; the value of *ai_protocol* shall be interpreted as in a call to the *socket*( ) function with the corresponding values of *ai_family* and *ai_protocol*.

RETURN VALUE

A zero return value for *getaddrinfo*( ) indicates successful completion; a non-zero return value indicates failure. The possible values for the failures are listed in the ERRORS section.

Upon successful return of *getaddrinfo*( ), the location to which *res* points shall refer to a linked list of **addrinfo** structures, each of which shall specify a socket address and information for use in creating a socket with which to use that socket address. The list shall include at least one **addrinfo** structure. The *ai_next* field of each structure contains a pointer to the next structure on the list, or a null pointer if it is the last structure on the list. Each structure on the list shall include values for use with a call to the *socket*( ) function, and a socket address for use with the *connect*( ) function or, if the AI_PASSIVE flag was specified, for use with the *bind*( ) function. The fields *ai_family*, *ai_socktype*, and *ai_protocol* shall be usable as the arguments to the *socket*( ) function to create a socket suitable for use with the returned address. The fields *ai_addr* and *ai_addrlen* are usable as the arguments to the *connect*( ) or *bind*( ) functions with such a socket, according to the AI_PASSIVE flag.

If *nodename* is not null, and if requested by the AI_CANONNAME flag, the *ai_canonname* field of the first returned **addrinfo** structure shall point to a null-terminated string containing the canonical name corresponding to the input *nodename*; if the canonical name is not available, then *ai_canonname* shall refer to the *nodename* argument or a string with the same contents. The contents of the *ai_flags* field of the returned structures are undefined.

All fields in socket address structures returned by *getaddrinfo*( ) that are not filled in through an explicit argument (for example, *sin6_flowinfo*) shall be set to zero.

Note: This makes it easier to compare socket address structures.

ERRORS

The *getaddrinfo*( ) function shall fail and return the corresponding error value if:

[EAI_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

[EAI_BADFLAGS] The *flags* parameter had an invalid value.

[EAI_FAIL] A non-recoverable error occurred when attempting to resolve the name.

[EAI_FAMILY] The address family was not recognized.

[EAI_MEMORY] There was a memory allocation failure when trying to allocate storage for the return value.

[EAI_NONAME] The name does not resolve for the supplied parameters.

Neither *nodename* nor *servname* were supplied. At least one of these shall be supplied.

[EAI_SERVICE] The service passed was not recognized for the specified socket type.

[EAI_SOCKTYPE]
The intended socket type was not recognized.

[EAI_SYSTEM] A system error occurred; the error code can be found in *errno*.

EXAMPLES

The following (incomplete) program demonstrates the use of *getaddrinfo*( ) to obtain the socket address structure(s) for the service named in the program's command-line argument. The program then loops through each of the address structures attempting to create and bind a socket to the address, until it performs a successful *bind*( ).

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>

int
main(int argc, char *argv[])
{
    struct addrinfo *result, *rp;
    int sfd, s;

    if (argc != 2) {
        fprintf(stderr, "Usage: %s port\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    struct addrinfo hints = {0};
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_DGRAM;
    hints.ai_flags = AI_PASSIVE;
    hints.ai_protocol = 0;

    s = getaddrinfo(NULL, argv[1], &hints, &result);
    if (s != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
        exit(EXIT_FAILURE);
    }

    /* getaddrinfo() returns a list of address structures.
       Try each address until a successful bind().
       If socket(2) (or bind(2)) fails, close the socket
       and try the next address. */

    for (rp = result; rp != NULL; rp = rp->ai_next) {
        sfd = socket(rp->ai_family, rp->ai_socktype,
            rp->ai_protocol);
        if (sfd == -1)
            continue;

        if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
            break;            /* Success */

        close(sfd);
```



```
        }

        if (rp == NULL) {          /* No address succeeded */
            fprintf(stderr, "Could not bind\n");
            exit(EXIT_FAILURE);
        }

        freeaddrinfo(result);      /* No longer needed */

                /* ... use socket bound to sfd ... */
    }
```

APPLICATION USAGE

If the caller handles only TCP and not UDP, for example, then the *ai_protocol* member of the *hints* structure should be set to IPPROTO_TCP when *getaddrinfo*( ) is called.

If the caller handles only IPv4 and not IPv6, then the *ai_family* member of the *hints* structure should be set to AF_INET when *getaddrinfo*( ) is called.

The *hints* structure can be initialized using `memset(&hints, 0, sizeof hints)` or by default initialization (see the APPLICATION USAGE for XBD **<netdb.h>**).

The term ``canonical name'' is misleading; it is taken from the Domain Name System (RFC 2181). It should be noted that the canonical name is a result of alias processing, and not necessarily a unique attribute of a host, address, or set of addresses. See RFC 2181 for more discussion of this in the Domain Name System context.

The *ai_socktype* field pointed to by *hints* is just the socket type; not the socket type and flags that can be specified when the socket is created. Passing in socket creation flags will cause a failure with [EAI_SOCKTYPE].

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

connect( ), *endservent*( ), *gai_strerror*( ), *getnameinfo*( ), *socket*( )

XBD **<netdb.h>**, **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The **restrict** keyword is added to the *getaddrinfo*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/19 is applied, adding three notes to the DESCRIPTION and adding text to the APPLICATION USAGE related to the term ``canonical name''. A reference to RFC 2181 is also added to the Informative References.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/20 is applied, making changes for alignment with IPv6. These include the following:

- Adding AI_V4MAPPED, AI_ALL, and AI_ADDRCONFIG to the allowed values for the *ai_flags* field
- Adding a description of AI_ADDRCONFIG

35205 • Adding a description of the consequences of ignoring the AI_PASSIVE flag.

35206 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/39 is applied, changing ``corresponding
35207 value'' to ``corresponding error value'' in the ERRORS section.

35208 **Issue 7**

35209 Austin Group Interpretation 1003.1-2001 #013 is applied.

35210 Austin Group Interpretation 1003.1-2001 #146 is applied, updating the DESCRIPTION.

35211 An example is added.

35212 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0130 [939], XSH/TC2-2008/0131 [979],
35213 XSH/TC2-2008/0132 [918], and XSH/TC2-2008/0133 [934] are applied.

35214 **Issue 8**

35215 Austin Group Defect 385 is applied, adding a requirement that *freeaddrinfo*( ) does not modify
35216 *errno* when passed a sublist that can be freed.

35217 Austin Group Defect 411 is applied, changing the ``socket type'' reference and adding a
35218 paragraph about *hints->ai_socktype* to the APPLICATION USAGE section.

35219 Austin Group Defect 940 is applied, changing text in the APPLICATION USAGE section relating
35220 to initialization of the *hints* structure.

35221 Austin Group Defect 1102 is applied, replacing a reference to the *inet_addr*( ) page with one to the
35222 *inet_ntop*( ) page.

35223 Austin Group Defect 1685 is applied, updating RFC references.

NAME

freelocale — free resources allocated for a locale object

SYNOPSIS

CX
```
#include <locale.h>

void freelocale(locale_t locobj);
```

DESCRIPTION

The *freelocale*( ) function shall cause the resources allocated for a locale object returned by a call to the *newlocale*( ) or *duplocale*( ) functions to be released. The *freelocale*( ) function shall not modify *errno* if *locobj* is a locale object previously returned by *newlocale*( ) or *duplocale*( ) and not yet released by *freelocale*( ) or *newlocale*( ).

The behavior is undefined if the *locobj* argument is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

Any use of a locale object that has been freed results in undefined behavior.

RETURN VALUE

None.

ERRORS

None.

EXAMPLES

Freeing Up a Locale Object

The following example shows a code fragment to free a locale object created by *newlocale*( ):

```
#include <locale.h>
...

/* Every locale object allocated with newlocale() should be
 * freed using freelocale():
 */

locale_t loc;

/* Get the locale. */

loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", NULL);

/* ... Use the locale object ... */
...

/* Free the locale object resources. */
freelocale (loc);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

duplocale( ), *getlocalename_l*( ), *newlocale*( ), *uselocale*( )

XBD **<locale.h>**

CHANGE HISTORY

First released in Issue 7.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0180 [283] is applied.

Issue 8

Austin Group Defect 385 is applied, adding a requirement that *freelocale*( ) does not modify *errno* when passed a locale object than can be freed.

Austin Group Defect 1220 is applied, adding *getlocalename_l*( ) to the SEE ALSO section.

NAME

freopen — open a stream

SYNOPSIS

```
#include <stdio.h>

FILE *freopen(const char *restrict pathname, const char *restrict mode,
    FILE *restrict stream);
```

DESCRIPTION

CX Except for the ``exclusive access'' requirement (see *fopen( )*), the functionality described on this reference page is aligned with the ISO C standard. Any other conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard for all *freopen*( ) functionality except in relation to ``exclusive access''.

The *freopen*( ) function shall first attempt to flush the stream associated with *stream* as if by a call to *fflush*(*stream*). Failure to flush the stream successfully shall be ignored. If *pathname* is not a null pointer, *freopen*( ) shall close any file descriptor associated with *stream*. Failure to close the file descriptor successfully shall be ignored. The error and end-of-file indicators for the stream shall be cleared.

The *freopen*( ) function shall open the file whose pathname is the string pointed to by *pathname* and associate the stream pointed to by *stream* with it. The *mode* argument shall be used just as in *fopen*( ).

The original stream shall be closed regardless of whether the subsequent open succeeds.

If *pathname* is a null pointer, the *freopen*( ) function shall attempt to change the mode of the stream to that specified by *mode*, as if the name of the file currently associated with the stream had been used. In this case, the file descriptor associated with the stream need not be closed if the call to *freopen*( ) succeeds. It is implementation-defined which changes of mode are permitted (if any), and under what circumstances.

XSI After a successful call to the *freopen*( ) function, the orientation of the stream shall be cleared, the encoding rule shall be cleared, and the associated **mbstate_t** object shall be set to describe an initial conversion state.

CX If *pathname* is not a null pointer, or if *pathname* is a null pointer and the specified mode change necessitates the file descriptor associated with the stream to be closed and reopened, the file descriptor associated with the reopened stream shall be allocated and opened as if by a call to *open*( ) with the flags specified for *fopen*( ) with the same *mode* argument.

RETURN VALUE

Upon successful completion, *freopen*( ) shall return the value of *stream*. Otherwise, a null pointer shall be returned, CX and *errno* shall be set to indicate the error.

ERRORS

The *freopen*( ) function shall fail if:

CX [EACCES] Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by *mode* are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created.

CX [EBADF] The file descriptor underlying the stream is not a valid file descriptor when *pathname* is a null pointer.

CX [EILSEQ] The *mode* argument begins with *w* or *a*, the file did not previously exist, and the last pathname component is not a portable filename and cannot be created in the target directory.

CX [EEXIST] The *mode* argument begins with *w* or *a* and includes *x*, but the file already exists.

CX [EINTR] A signal was caught during *freopen*( ).

CX [EISDIR] The named file is a directory and *mode* requires write access.

CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *pathname* argument.

CX [EMFILE] All file descriptors available to the process are currently open.

CX [ENAMETOOLONG]
The length of a component of a pathname is longer than {NAME_MAX}.

CX [ENFILE] The maximum allowable number of files is currently open in the system.

CX [ENOENT] The *mode* string begins with `'r'` and a component of *pathname* does not name an existing file, or *mode* begins with `'w'` or `'a'` and a component of the path prefix of *pathname* does not name an existing file, or *pathname* is an empty string.

CX [ENOENT] or [ENOTDIR]
The *pathname* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters. If *pathname* without the trailing <slash> characters would name an existing file, an [ENOENT] error shall not occur.

CX [ENOSPC] The directory or file system that would contain the new file cannot be expanded, the file does not exist, and it was to be created.

CX [ENOTDIR] A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *pathname* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

CX [ENXIO] The named file is a character special or block special file, and the device associated with this special file does not exist.

CX [EOVERFLOW] The named file is a regular file and the size of the file cannot be represented correctly in an object of type **off_t**.

CX [EROFS] The named file resides on a read-only file system and *mode* requires write access.

The *freopen*( ) function may fail if:

CX [EBADF] The mode with which the file descriptor underlying the stream was opened does not support the requested mode when *pathname* is a null pointer.

CX [EINVAL] The value of the *mode* argument is not valid.

CX [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *pathname* argument.

CX [ENAMETOOLONG] The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

CX [ENOMEM] Insufficient storage space is available.

CX [ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.

CX [ETXTBSY] The file is a pure procedure (shared text) file that is being executed and *mode* requires write access.

EXAMPLES

Directing Standard Output to a File

The following example logs all standard output to the **/tmp/logfile** file.

```
#include <stdio.h>
...
FILE *fp;
...
fp = freopen ("/tmp/logfile", "a+", stdout);
...
```

APPLICATION USAGE

The *freopen*( ) function is typically used to attach the pre-opened *streams* associated with *stdin*, *stdout*, and *stderr* to other files.

Since implementations are not required to support any stream mode changes when the *pathname* argument is NULL, portable applications cannot rely on the use of *freopen*( ) to change the stream mode, and use of this feature is discouraged. The feature was originally added to the ISO C standard in order to facilitate changing *stdin* and *stdout* to binary mode. Since a `'b'` character in the mode has no effect on POSIX systems, this use of the feature is unnecessary in POSIX applications. However, even though the `'b'` is ignored, a successful call to *freopen*(NULL, "*wb*", *stdout*) does have an effect. In particular, for regular files it truncates the file and sets the file-position indicator for the stream to the start of the file. It is possible that these side-effects are an unintended consequence of the way the feature was specified in the ISO/IEC 9899:1999 standard (and still is in the current standard), but unless or until the ISO C standard is changed, applications which successfully call *freopen*(NULL, "*wb*", *stdout*) will behave in unexpected ways on conforming systems in situations such as:

```
{ appl file1; appl file2; } > file3
```

which will result in **file3** containing only the output from the second invocation of *appl*.

See also the APPLICATION USAGE for *fopen*( ).

RATIONALE

See the RATIONALE for *fopen*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fclose*( ), *fdopen*( ), *fflush*( ), *fmemopen*( ), *fopen*( ), *mbsinit*( ), *open*( ), *open_memstream*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate that the orientation of the stream is cleared and the conversion state of the stream is set to an initial conversion state by a successful call to the *freopen*( ) function.

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files.
- In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support large files.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [EINVAL], [ENOMEM], [ENXIO], and [ETXTBSY] optional error conditions are added.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *freopen*( ) prototype is updated.
- The DESCRIPTION is updated.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

The DESCRIPTION is updated regarding failure to close, changing the ``file'' to ``file descriptor''.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/40 is applied, adding the following sentence to the DESCRIPTION: ``In this case, the file descriptor associated with the stream need not be closed if the call to *freopen*( ) succeeds.''.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/41 is applied, adding an mandatory [EBADF] error, and an optional [EBADF] error to the ERRORS section.

Issue 7

Austin Group Interpretation 1003.1-2001 #043 is applied, clarifying that the *freopen*( ) function allocates a file descriptor as per *open*( ).

Austin Group Interpretation 1003.1-2001 #143 is applied.

Austin Group Interpretation 1003.1-2001 #159 is applied, clarifying requirements for the flags set on the open file description.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-150 and SD5-XSH-ERN-219 are applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0181 [291,433], XSH/TC1-2008/0182 [146,433], XSH/TC1-2008/0183 [324], and XSH/TC1-2008/0184 [14] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0134 [822] is applied.

Issue 8

Austin Group Defect 293 is applied, adding the [EILSEQ] error.

Austin Group Defect 411 is applied, adding the *e* and *x* mode string characters.

Austin Group Defect 1200 is applied, correcting the argument name in the [ELOOP] errors.

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018 standard.

NAME

frexp, frexpf, frexpl — extract significand and exponent from a double precision number

SYNOPSIS

```
#include <math.h>

double frexp(double num, int *exp);
float frexpf(float num, int *exp);
long double frexpl(long double num, int *exp);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall break a floating-point number *num* into a normalized fraction and an integral power of 2. The integer exponent shall be stored in the **int** object pointed to by *exp*; if the integer exponent is outside the range of **int**, the results are unspecified.

RETURN VALUE

For finite arguments, these functions shall return the value *x*, such that *x* has a magnitude in the interval [½,1) or 0, and *num* equals *x* times 2 raised to the power **exp*.

MX When the radix of the argument is a power of 2, the returned value shall be exact and shall be independent of the current rounding direction mode.

If *num* is NaN, a NaN shall be returned, and the value of **exp* is unspecified.

If *num* is ±0, ±0 shall be returned, and the value of **exp* shall be 0.

If *num* is ±Inf, *num* shall be returned, and the value of **exp* is unspecified.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isnan( ), *ldexp*( ), *modf*( )

XBD **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *frexpf*( ) and *frexpl*( ) functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

Austin Group Defect 1753 is applied, changing the NAME section.

NAME

fscanf, scanf, sscanf — convert formatted input

SYNOPSIS

```
#include <stdio.h>

int fscanf(FILE *restrict stream, const char *restrict format, ...);
int scanf(const char *restrict format, ...);
int sscanf(const char *restrict s, const char *restrict format, ...);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fscanf*( ) function shall read from the named input *stream*. The *scanf*( ) function shall read from the standard input stream *stdin*. The *sscanf*( ) function shall read from the string *s*. Each function reads bytes, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string *format* described below, and a set of *pointer* arguments indicating where the converted input should be stored. The result is undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments shall be evaluated but otherwise ignored.

CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than to the next unused argument. In this case, the conversion specifier character `%` (see below) is replaced by the sequence `"%`*n*`$"`, where *n* is a decimal integer in the range [1,{NL_ARGMAX}]. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages. In format strings containing the `"%`*n*`$"` form of conversion specifications, it is unspecified whether numbered arguments in the argument list can be referenced from the format string more than once.

The format can contain either form of a conversion specification—that is, `%` or `"%`*n*`$"`—but the two forms cannot be mixed within a single format string. The only exception to this is that `%%` or `%*` can be mixed with the `"%`*n*`$"` form. When numbered argument specifications are used, specifying the *N*th argument requires that all the leading arguments, from the first to the (*N*−1)th, are pointers.

The *fscanf*( ) function in all its forms shall allow detection of a language-dependent radix character in the input string. The radix character is defined in the current locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a <period> (`'.'`).

The application shall ensure that the format is a character string, beginning and ending in its initial shift state, if any, composed of zero or more directives. Each directive is composed of one of the following: one or more white-space bytes; an ordinary character (neither `'%'` nor a white-space byte); or a conversion specification. Each conversion specification is introduced by the character `'%'` CX or the character sequence `"%`*n*`$"`, after which the following appear in sequence:

- An optional assignment-suppressing character `'*'`.
- An optional non-zero decimal integer that specifies the maximum field width.
- CX An optional assignment-allocation character `'m'`.
- An option length modifier that specifies the size of the receiving object.
- A *conversion specifier* character that specifies the type of conversion to be applied. The valid conversion specifiers are described below.

The *fscanf*( ) functions shall execute each directive of the format in turn. When all directives have been executed, or if a directive fails (as detailed below), the function shall return. Failures are described as input failures (due to the unavailability of input bytes) or matching failures (due to inappropriate input).

A directive composed of one or more white-space bytes shall be executed by reading input up to the first non-white-space byte, which shall remain unread, or until no more bytes can be read. The directive shall never fail.

A directive that is an ordinary character shall be executed as follows: the next byte shall be read from the input and compared with the byte that comprises the directive; if the comparison shows that they are not equivalent, the directive shall fail, and the differing and subsequent bytes shall remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a character from being read, the directive shall fail.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each conversion character. A conversion specification shall be executed in the following steps.

Input white-space bytes shall be skipped, unless the conversion specification includes a `[`, `c`, `C`, or `n` conversion specifier.

An item shall be read from the input, unless the conversion specification includes an `n` conversion specifier. An input item shall be defined as the longest sequence of input bytes (up to any specified maximum field width, which may be measured in characters or bytes dependent on the conversion specifier) which is an initial subsequence of a matching sequence. The first byte, if any, after the input item shall remain unread. If the length of the input item is 0, the execution of the conversion specification shall fail; this condition is a matching failure, unless end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is an input failure.

Except in the case of a `%` conversion specifier, the input item (or, in the case of a `%n` conversion specification, the count of input bytes) shall be converted to a type appropriate to the conversion character. If the input item is not a matching sequence, the execution of the conversion specification fails; this condition is a matching failure. Unless assignment suppression was indicated by a `'*'`, the result of the conversion shall be placed in the object pointed to by the first argument following the *format* argument that has not already received a conversion result if
CX the conversion specification is introduced by `%`, or in the *n*th argument if introduced by the character sequence `"%`*n*`$"`. If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.

CX The `c`, `s`, and `[` conversion specifiers shall accept an optional assignment-allocation character `'m'`, which shall cause a memory buffer to be allocated to hold the conversion results. If the conversion specifier is `s` or `[`, the allocated buffer shall include space for a terminating null character (or wide character). In such a case, the argument corresponding to the conversion specifier should be a reference to a pointer variable that will receive a pointer to the allocated buffer. The system shall allocate a buffer as if *malloc*( ) had been called. The application shall be responsible for freeing the memory after usage. If there is insufficient memory to allocate a buffer, the function shall set *errno* to [ENOMEM] and a conversion error shall result. If the function returns EOF, any memory successfully allocated for parameters using assignment-allocation character `'m'` by this call shall be freed before the function returns.

The length modifiers and their meanings are:

`hh` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **signed char** or **unsigned char**.

`h` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **short** or **unsigned short**.

`l` (ell) Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **long** or **unsigned long**; that a following `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier applies to an argument with type pointer to **double**; or that a following `c`, `s`, or `[` conversion specifier applies to an argument with type pointer to **wchar_t**. CX If the `'m'` assignment-allocation character is specified, the conversion applies to an argument with the type pointer to a pointer to **wchar_t**.

`ll` (ell-ell) Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **long long** or **unsigned long long**.

`j` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **intmax_t** or **uintmax_t**.

`z` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **size_t** or the corresponding signed integer type.

`t` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **ptrdiff_t** or the corresponding **unsigned** type.

`L` Specifies that a following `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier applies to an argument with type pointer to **long double**.

If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.

The following conversion specifiers are valid:

`d` Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of *strtol*() with the value 10 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **int**.

`i` Matches an optionally signed integer, whose format is the same as expected for the subject sequence of *strtol*() with 0 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **int**.

`o` Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of *strtoul*() with the value 8 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **unsigned**.

`u` Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of *strtoul*() with the value 10 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **unsigned**.

`x` Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of *strtoul*() with the value 16 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding

35630 argument is a pointer to **unsigned**.

35631 `a`, `e`, `f`, `g`

35632 Matches an optionally signed floating-point number, infinity, or NaN, whose format is
35633 the same as expected for the subject sequence of *strtod*( ). In the absence of a size
35634 modifier, the application shall ensure that the corresponding argument is a pointer to
35635 **float**.

35636 If the *fprintf*( ) family of functions generates character string representations for infinity
35637 and NaN (a symbolic entity encoded in floating-point format) to support
35638 IEEE Std 754-1985, the *fscanf*( ) family of functions shall recognize them as input.

35639 `s` Matches a sequence of bytes that are not white-space bytes. If the `'m'` assignment-
35640 allocation character is not specified, the application shall ensure that the corresponding
35641 argument is a pointer to the initial byte of an array of **char**, **signed char**, or **unsigned**
35642 **char** large enough to accept the sequence and a terminating null character code, which
35643 CX shall be added automatically. Otherwise, the application shall ensure that the
35644 corresponding argument is a pointer to a pointer to a **char**.

35645 If an `l` (ell) qualifier is present, the input is a sequence of characters that begins in the
35646 initial shift state. Each character shall be converted to a wide character as if by a call to
35647 the *mbrtowc*( ) function, with the conversion state described by an **mbstate_t** object
35648 initialized to zero before the first character is converted. If the `'m'` assignment-
35649 allocation character is not specified, the application shall ensure that the corresponding
35650 argument is a pointer to an array of **wchar_t** large enough to accept the sequence and
35651 CX the terminating null wide character, which shall be added automatically. Otherwise,
35652 the application shall ensure that the corresponding argument is a pointer to a pointer to
35653 a **wchar_t**.

35654 `[` Matches a non-empty sequence of bytes from a set of expected bytes (the *scanset*). The
35655 normal skip over white-space bytes shall be suppressed in this case. If the `'m'`
35656 assignment-allocation character is not specified, the application shall ensure that the
35657 corresponding argument is a pointer to the initial byte of an array of **char**, **signed char**,
35658 or **unsigned char** large enough to accept the sequence and a terminating null byte,
35659 CX which shall be added automatically. Otherwise, the application shall ensure that the
35660 corresponding argument is a pointer to a pointer to a **char**.

35661 If an `l` (ell) qualifier is present, the input is a sequence of characters that begins in the
35662 initial shift state. Each character in the sequence shall be converted to a wide character
35663 as if by a call to the *mbrtowc*( ) function, with the conversion state described by an
35664 **mbstate_t** object initialized to zero before the first character is converted. If the `'m'`
35665 assignment-allocation character is not specified, the application shall ensure that the
35666 corresponding argument is a pointer to an array of **wchar_t** large enough to accept the
35667 sequence and the terminating null wide character, which shall be added automatically.
35668 CX Otherwise, the application shall ensure that the corresponding argument is a pointer to
35669 a pointer to a **wchar_t**.

35670 The conversion specification includes all subsequent bytes in the format string up to
35671 and including the matching <right-square-bracket> (`']'`). The bytes between the
35672 square brackets (the *scanlist*) comprise the scanset, unless the byte after the <left-
35673 square-bracket> is a <circumflex> (`'^'`), in which case the scanset contains all bytes
35674 that do not appear in the scanlist between the <circumflex> and the <right-square-
35675 bracket>. If the conversion specification begins with `"[]"` or `"[^]"`, the <right-
35676 square-bracket> is included in the scanlist and the next <right-square-bracket> is the
35677 matching <right-square-bracket> that ends the conversion specification; otherwise, the

first <right-square-bracket> is the one that ends the conversion specification. If a '−' is in the scanlist and is not the first character, nor the second where the first character is a '^', nor the last character, the behavior is implementation-defined.

c Matches a sequence of bytes of the number specified by the field width (1 if no field width is present in the conversion specification). No null byte is added. The normal skip over white-space bytes shall be suppressed in this case. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to the initial byte of an array of **char**, **signed char**, or **unsigned char** large enough to accept the sequence. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a **char**.

If an l (ell) qualifier is present, the input shall be a sequence of characters that begins in the initial shift state. Each character in the sequence is converted to a wide character as if by a call to the *mbrtowc*() function, with the conversion state described by an **mbstate_t** object initialized to zero before the first character is converted. No null wide character is added. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of **wchar_t** large enough to accept the resulting sequence of wide characters. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a **wchar_t**.

p Matches an implementation-defined set of sequences, which shall be the same as the set of sequences that is produced by the %p conversion specification of the corresponding *fprintf*() functions. The application shall ensure that the corresponding argument is a pointer to a pointer to **void**. The interpretation of the input item is implementation-defined. If the input item is a value converted earlier during the same program execution, the pointer that results shall compare equal to that value; otherwise, the behavior of the %p conversion specification is undefined.

n No input is consumed. The application shall ensure that the corresponding argument is a pointer to the integer into which shall be written the number of bytes read from the input so far by this call to the *fscanf*() functions. Execution of a %n conversion specification shall not increment the assignment count returned at the completion of execution of the function. No argument shall be converted, but one shall be consumed. If the conversion specification includes an assignment-suppressing character or a field width, the behavior is undefined.

XSI C Equivalent to lc.

XSI S Equivalent to ls.

% Matches a single '%' character; no conversion or assignment occurs. The complete conversion specification shall be %%.

If a conversion specification is invalid, the behavior is undefined.

The conversion specifiers A, E, F, G, and X are also valid and shall be equivalent to a, e, f, g, and x, respectively.

If end-of-file is encountered during input, conversion shall be terminated. If end-of-file occurs before any bytes matching the current conversion specification (except for %n) have been read (other than leading white-space bytes, where permitted), execution of the current conversion specification shall terminate with an input failure. Otherwise, unless execution of the current conversion specification is terminated with a matching failure, execution of the following conversion specification (if any) shall be terminated with an input failure.

Reaching the end of the string in *sscanf*( ) shall be equivalent to encountering end-of-file for *fscanf*( ).

If conversion terminates on a conflicting input, the offending input is left unread in the input. Any trailing white-space bytes (including <newline> characters) shall be left unread unless matched by a conversion specification. The success of literal matches and suppressed assignments is only directly determinable via the `%n` conversion specification.

CX The *fscanf*( ) and *scanf*( ) functions may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetc*( ), *fgets*( ), *fread*( ), *getc*( ), *getchar*( ), *getdelim*( ), *getline*( ), *fscanf*( ), or *scanf*( ) using *stream* that returns data not supplied by a prior call to *ungetc*( ).

RETURN VALUE

Upon successful completion, these functions shall return the number of successfully matched and assigned input items; this number can be zero in the event of an early matching failure. If the input ends before the first conversion (if any) has completed, and without a matching failure having occurred, EOF shall be returned. If an error occurs before the first conversion (if any) has
CX completed, and without a matching failure having occurred, EOF shall be returned and *errno* shall be set to indicate the error. If an error occurs, the error indicator for the stream shall be set.

ERRORS

For the conditions under which the *fscanf*( ) functions fail and may fail, refer to *fgetc*( ) or *fgetwc*( ).

In addition, the *fscanf*( ) function shall fail if:

CX [EILSEQ] Input byte sequence does not form a valid character.

[ENOMEM] Insufficient storage space is available.

In addition, the *fscanf*( ) function may fail if:

CX [EINVAL] There are insufficient arguments.

EXAMPLES

The call:

```
int i, n; float x; char name[50];
n = scanf("%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 Hamster
```

assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string `"Hamster"`.

The call:

```
int i; float x; char name[50];
(void) scanf("%2d%f%*d %[0123456789]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```

assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string `"56\0"` in *name*. The next call to *getchar*( ) shall return the character `'a'`.

35764 **Reading Data into an Array**

35765 The following call uses *fscanf*( ) to read three floating-point numbers from standard input into
35766 the *input* array.

35767 ```
float input[3]; fscanf (stdin, "%f %f %f", input, input+1, input+2);
```

35768 **APPLICATION USAGE**

35769 If the application calling *fscanf*( ) has any objects of type **wint_t** or **wchar_t**, it must also include
35770 the **<wchar.h>** header to have these objects defined.

35771 For functions that allocate memory as if by *malloc*( ), the application should release such memory
35772 when it is no longer required by a call to *free*( ). For *fscanf*( ), this is memory allocated via use of
35773 the `'m'` assignment-allocation character.

35774 **RATIONALE**

35775 The set of characters allowed in a scanset is limited to single-byte characters. In other similar
35776 places, multi-byte characters have been permitted, but for alignment with the ISO C standard, it
35777 has not been done here. Applications needing this could use the corresponding wide-character
35778 functions to achieve the desired results.

35779 **FUTURE DIRECTIONS**

35780 None.

35781 **SEE ALSO**

35782 Section 2.5 (on page 521), *fprintf*( ), *getc*( ), *setlocale*( ), *strtod*( ), *strtol*( ), *strtoul*( ), *wcrtomb*( )

35783 XBD Chapter 7 (on page 127), **<inttypes.h>**, **<langinfo.h>**, **<stdio.h>**, **<wchar.h>**

35784 **CHANGE HISTORY**

35785 First released in Issue 1. Derived from Issue 1 of the SVID.

35786 **Issue 5**

35787 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the `l` (ell) qualifier is
35788 now defined for the `c`, `s`, and `[` conversion specifiers.

35789 The DESCRIPTION is updated to indicate that if infinity and NaN can be generated by the
35790 *fprintf*( ) family of functions, then they are recognized by the *fscanf*( ) family.

35791 **Issue 6**

35792 The Open Group Corrigenda U021/7 and U028/10 are applied. These correct several
35793 occurrences of ``characters'' in the text which have been replaced with the term ``bytes''.

35794 The normative text is updated to avoid use of the term ``must'' for application requirements.

35795 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

35796 - The prototypes for *fscanf*( ), *scanf*( ), and *sscanf*( ) are updated.
35797 - The DESCRIPTION is updated.
35798 - The `hh`, `ll`, `j`, `t`, and `z` length modifiers are added.
35799 - The `a`, `A`, and `F` conversion characters are added.

35800 The DESCRIPTION is updated to use the terms ``conversion specifier'' and ``conversion
35801 specification'' consistently.

  
```

Issue 7

Austin Group Interpretation 1003.1-2001 #170 is applied.

SD5-XSH-ERN-9 is applied, correcting *fscanf*( ) to *scanf*( ) in the DESCRIPTION.

SD5-XSH-ERN-132 is applied, adding the assignment-allocation character `'m'`.

Functionality relating to the `%n$` form of conversion specification is moved from the XSI option to the Base.

Changes are made related to support for finegrained timestamps.

The APPLICATION USAGE section is updated to clarify that memory is allocated as if by *malloc*( ).

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0185 [302], XSH/TC1-2008/0186 [90], and XSH/TC1-2008/0187 [14] are applied. XSH/TC1-2008/0186 [90] changes the second sentence in the RETURN VALUE section to align with expected wording changes in the next revision of the ISO C standard.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0135 [936] is applied.

Issue 8

Austin Group Defect 1163 is applied, clarifying the handling of white space in the format string.

Austin Group Defect 1173 is applied, clarifying the description of the assignment-allocation character `'m'`.

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018 standard.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1375 is applied, changing ``terminating null character'' to ``terminating null character (or wide character)''.

Austin Group Defect 1562 is applied, clarifying that it is the application's responsibility to ensure that the format is a character string, beginning and ending in its initial shift state, if any.

Austin Group Defect 1624 is applied, changing the RETURN VALUE section.

NAME

fseek, fseeko — reposition a file-position indicator in a stream

SYNOPSIS

```
#include <stdio.h>

int fseek(FILE *stream, long offset, int whence);
CX int fseeko(FILE *stream, off_t offset, int whence);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fseek*( ) function shall set the file-position indicator for the stream pointed to by *stream*. If a read or write error occurs, the error indicator for the stream shall be set and *fseek*( ) fails.

CX The new position, measured in bytes from the beginning of the file, except in the case of streams opened with *open_wmemstream*( ) for which the position shall be measured in wide characters, shall be obtained by adding *offset* to the position specified by *whence*. The specified point is the beginning of the file for SEEK_SET, the current value of the file-position indicator for SEEK_CUR, or end-of-file for SEEK_END.

If the stream is to be used with wide-character input/output functions, the application shall ensure that *offset* is either 0 or a value returned by an earlier call to *ftell*( ) on the same stream and *whence* is SEEK_SET.

A successful call to *fseek*( ) shall clear the end-of-file indicator for the stream and undo any effects of *ungetc*( ) and *ungetwc*( ) on the same stream. After an *fseek*( ) call, the next operation on an update stream may be either input or output.

CX If the most recent operation, other than *ftell*( ), on a given stream is *fflush*( ), the file offset in the underlying open file description shall be adjusted to reflect the location specified by *fseek*( ).

The *fseek*( ) function shall allow the file-position indicator to be set beyond the end of existing data in the file. If data is later written at this point, subsequent reads of data in the gap shall return bytes with the value 0 until data is actually written into the gap.

The behavior of *fseek*( ) on devices which are incapable of seeking is implementation-defined. The value of the file offset associated with such a device is undefined.

If the stream has an underlying file description and is writable, and buffered data had not been written to the underlying file, *fseek*( ) shall cause the unwritten data to be written to the file and shall mark the last data modification and last file status change timestamps of the file for update. If the stream was created by *fmemopen*( ), *open_memstream*( ), or *open_wmemstream*( ) and the stream is writable, and if the stream is buffered and data in the stream's buffer has not been written to the underlying memory buffer, *fseek*( ) shall cause the unwritten data to be written to the underlying memory buffer.

In a locale with state-dependent encoding, whether *fseek*( ) restores the stream's shift state is implementation-defined.

The *fseeko*( ) function shall be equivalent to the *fseek*( ) function except that the *offset* argument is of type **off_t**.

35869 **RETURN VALUE**

35870 CX The *fseek*( ) and *fseeko*( ) functions shall return 0 if they succeed.

35871 CX Otherwise, they shall return –1 and set *errno* to indicate the error.

35872 **ERRORS**

35873 CX The *fseek*( ) and *fseeko*( ) functions shall fail if, either the *stream* is unbuffered or the *stream*'s
35874 buffer needed to be flushed, and the call to *fseek*( ) or *fseeko*( ) causes an underlying *lseek*( ) or
35875 *write*( ) to be invoked, and:

35876 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor and the thread would be
35877 delayed in the write operation.

35878 CX [EBADF] The file descriptor underlying the stream file is not open for writing or the
35879 stream's buffer needed to be flushed and the file is not open.

35880 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

35881 CX [EFBIG] An attempt was made to write a file that exceeds the file size limit of the
35882 process.
35883 XSI A SIGXFSZ signal shall also be generated for the thread.

35884 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
35885 offset maximum associated with the corresponding stream.

35886 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
35887 was transferred.

35888 CX [EINVAL] The *whence* argument is invalid, the resulting file-position indicator would be
35889 set to a negative value, or the stream was created by *fmemopen*( ) and the
35890 resulting file-position indicator would be beyond the end of the underlying
35891 memory buffer.

35892 CX [EIO] A physical I/O error has occurred, or the process is a member of a background
35893 process group attempting to perform a *write*( ) to its controlling terminal,
35894 TOSTOP is set, the calling thread is not blocking SIGTTOU, the process is not
35895 ignoring SIGTTOU, and the process group of the process is orphaned. This
35896 error may also be returned under implementation-defined conditions.

35897 CX [ENOMEM] The stream was created by *open_memstream*( ) or *open_wmemstream*( ) and
35898 insufficient memory is available.

35899 CX [ENOSPC] There was no free space remaining on the device containing the file or in the
35900 buffer used by the *fmemopen*( ) function.

35901 CX [EOVERFLOW] For *fseek*( ), the resulting file offset would be a value which cannot be
35902 represented correctly in an object of type **long**.

35903 CX [EOVERFLOW] For *fseeko*( ), the resulting file offset would be a value which cannot be
35904 represented correctly in an object of type **off_t**.

35905 CX [EPIPE] An attempt was made to write to a pipe or FIFO that is not open for reading
35906 by any process; a SIGPIPE signal shall also be sent to the thread.

35907 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.

35908 CX The *fseek*( ) and *fseeko*( ) functions may fail if:

CX [ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

When the stream was created by *fmemopen*( ), *fseek*( ) fails if an attempt is made to seek beyond the end of the underlying memory buffer. This is different than *fseek*( ) on a file when a file size limit is in effect because the size specified to *fmemopen*( ) is a fixed, absolute limit whereas a file size limit is artificial and can be changed. With a file size limit, it is possible to seek past the limit, then raise the limit and successfully write at the new position; there is no equivalent possibility with the buffer size specified to *fmemopen*( ).

See also the rationale for *ftell*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fopen*( ), *fmemopen*( ), *fsetpos*( ), *ftell*( ), *getrlimit*( ), *lseek*( ), *open_memstream*( ), *rewind*( ), *ungetc*( ), *write*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The *fseeko*( ) function is added.
- The [EFBIG], [EOVERFLOW], and [ENXIO] mandatory error conditions are added.

The following change is incorporated for alignment with the FIPS requirements:

- The [EINTR] error is no longer an indication that the implementation does not report partial transfers.

The normative text is updated to avoid use of the term ``must'' for application requirements.

The DESCRIPTION is updated to explicitly state that *fseek*( ) sets the file-position indicator, and then on error the error indicator is set and *fseek*( ) fails. This is for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/42 is applied, updating the [EAGAIN] error in the ERRORS section from ``the process would be delayed'' to ``the thread would be delayed''.

Issue 7

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0188 [79], XSH/TC1-2008/0189 [122], XSH/TC1-2008/0190 [225], and XSH/TC1-2008/0191 [14] are applied.

Issue 8

Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

Austin Group Defect 1027 is applied, specifying that for streams opened with *open_wmemstream*( ) the position is measured in wide characters, not bytes.

Austin Group Defect 1225 is applied, clarifying the behavior for memory streams.

Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error relating to the file size limit for the process.

NAME

fsetpos — set current file position

SYNOPSIS

```
#include <stdio.h>

int fsetpos(FILE *stream, const fpos_t *pos);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fsetpos*( ) function shall set the **mbstate_t** object (if any) and file position indicator for the stream pointed to by *stream* according to the value of the object pointed to by *pos*, which the application shall ensure is a value obtained from an earlier call to *fgetpos*( ) on the same stream. If a read or write error occurs, the error indicator for the stream shall be set and *fsetpos*( ) fails.

A successful call to the *fsetpos*( ) function shall undo any effects of the *ungetc*( ) function on the stream, clear the end-of-file indicator for the stream, and then establish the new parse state and position. After a successful *fsetpos*( ) call, the next operation on an update stream can be either input or output.

CX The behavior of *fsetpos*( ) on devices which are incapable of seeking is implementation-defined. The value of the file offset associated with such a device is undefined.

If the stream has an underlying file description and is writable, and buffered data has not been written to the underlying file, *fsetpos*( ) shall cause the unwritten data to be written to the file and shall mark the last data modification and last file status change timestamps of the file for update.

If the stream was created by *fmemopen*( ), *open_memstream*( ), or *open_wmemstream*( ) and the stream is writable, and if the stream is buffered and data in the stream's buffer has not been written to the underlying memory buffer, *fsetpos*( ) shall cause the unwritten data to be written to the underlying memory buffer.

The *fsetpos*( ) function shall not change the setting of *errno* if successful.

RETURN VALUE

The *fsetpos*( ) function shall return 0 if it succeeds; otherwise, it shall return a non-zero value and set *errno* to indicate the error.

ERRORS

CX The *fsetpos*( ) function shall fail if, either the *stream* is unbuffered or the *stream*'s buffer needed to be flushed, and the call to *fsetpos*( ) causes an underlying *lseek*( ) or *write*( ) to be invoked, and:

CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor and the thread would be delayed in the write operation.

CX [EBADF] The file descriptor underlying the stream file is not open for writing or the stream's buffer needed to be flushed and the file is not open.

CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

CX [EFBIG] An attempt was made to write a file that exceeds the file size limit of the process.

XSI A SIGXFSZ signal shall also be generated for the thread.

CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.

36005 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
36006 was transferred.

36007 CX [EIO] A physical I/O error has occurred, or the process is a member of a background
36008 process group attempting to perform a *write*( ) to its controlling terminal,
36009 TOSTOP is set, the calling thread is not blocking SIGTTOU, the process is not
36010 ignoring SIGTTOU, and the process group of the process is orphaned. This
36011 error may also be returned under implementation-defined conditions.

36012 CX [ENOMEM] The stream was created by *open_memstream*( ) or *open_wmemstream*( ) and
36013 insufficient memory is available.

36014 CX [ENOSPC] There was no free space remaining on the device containing the file or in the
36015 buffer used by the *fmemopen*( ) function.

36016 CX [EPIPE] An attempt was made to write to a pipe or FIFO that is not open for reading
36017 by any process; a SIGPIPE signal shall also be sent to the thread.

36018 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.

36019 The *fsetpos*( ) function may fail if:

36020 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
36021 capabilities of the device.

36022 **EXAMPLES**

36023 None.

36024 **APPLICATION USAGE**

36025 None.

36026 **RATIONALE**

36027 The ERRORS section does not include an [EINVAL] error equivalent to the one for *fseek*( )
36028 because applications are required to obtain the **fpos_t** value using *fgetpos*( ), in which case the
36029 file position to be set will always be valid. Directly manipulating the **fpos_t** object to set a
36030 position results in undefined behavior. However, if an implementation detects that the
36031 requested file position would be a negative value, or would be beyond the end of the underlying
36032 memory buffer of a stream that was created by *fmemopen*( ), it is recommended that *fsetpos*( )
36033 returns a non-zero value and sets errno to [EINVAL].

36034 **FUTURE DIRECTIONS**

36035 None.

36036 **SEE ALSO**

36037 Section 2.5 (on page 521), *fopen*( ), *fmemopen*( ), *ftell*( ), *lseek*( ), *open_memstream*( ), *rewind*( ),
36038 *ungetc*( ), *write*( )

36039 XBD **<stdio.h>**

36040 **CHANGE HISTORY**

36041 First released in Issue 4. Derived from the ISO C standard.

36042 **Issue 6**

36043 Extensions beyond the ISO C standard are marked.

36044 The normative text is updated to avoid use of the term ``must'' for application requirements.

36045 The DESCRIPTION is updated to clarify that the error indicator is set for the stream on a read or
36046 write error. This is for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/21 is applied, deleting an erroneous [EINVAL] error case from the ERRORS section.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/43 is applied, updating the [EAGAIN] error in the ERRORS section from ``the process would be delayed'' to ``the thread would be delayed''.

Issue 7

SD5-XSH-ERN-220 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0192 [105], XSH/TC1-2008/0193 [79], XSH/TC1-2008/0194 [225], XSH/TC1-2008/0195 [450], XSH/TC1-2008/0196 [450], and XSH/TC1-2008/0197 [14] are applied.

Issue 8

Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

Austin Group Defect 1225 is applied, aligning the CX requirements with *fseek*( ).

Austin Group Defect 1249 is applied, correcting some text mismatches with the ISO C standard.

Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error relating to the file size limit for the process.

NAME

fstat — get file status

SYNOPSIS

```
#include <sys/stat.h>

int fstat(int fildes, struct stat *buf);
```

DESCRIPTION

The *fstat*( ) function shall obtain information about an open file associated with the file descriptor *fildes*, and shall write it to the area pointed to by *buf*.

SHM If *fildes* references a shared memory object, the implementation shall update in the **stat** structure pointed to by the *buf* argument the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be valid. The implementation may update other fields and flags.

TYM If *fildes* references a typed memory object, the implementation shall update in the **stat** structure pointed to by the *buf* argument the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be valid. The implementation may update other fields and flags.

The *buf* argument is a pointer to a **stat** structure, as defined in **<sys/stat.h>**, into which information is placed concerning the file.

For all other file types defined in this volume of POSIX.1-2024, the structure members *st_mode*, *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atim*, *st_ctim*, and *st_mtim* shall have meaningful values and the value of the *st_nlink* member shall be set to the number of links to the file.

An implementation that provides additional or alternative file access control mechanisms may, under implementation-defined conditions, cause *fstat*( ) to fail.

The *fstat*( ) function shall update any time-related fields (as described in XBD Section 4.12, on page 98), before writing into the **stat** structure.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *fstat*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

[EIO] An I/O error occurred while reading from the file system.

[EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file serial number cannot be represented correctly in the structure pointed to by *buf*.

The *fstat*( ) function may fail if:

[EOVERFLOW] One of the values is too large to store into the structure pointed to by the *buf* argument.

EXAMPLES

Obtaining File Status Information

The following example shows how to obtain file status information for a file named **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure. The **/home/cnd/mod1** file is opened with read/write privileges and is passed to the open file descriptor *fildes*.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

struct stat buffer;
int         status;
...
fildes = open("/home/cnd/mod1", O_RDWR);
status = fstat(fildes, &buffer);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fstatat( )

XBD Section 4.12 (on page 98), **<sys/stat.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

Large File Summit extensions are added.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [EIO] mandatory error condition is added.
- The [EOVERFLOW] mandatory error condition is added. This change is to support large files.
- The [EOVERFLOW] optional error condition is added.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that shared memory object semantics apply to typed memory objects.

Issue 7

XSH-SD5-ERN-161 is applied, updating the DESCRIPTION to clarify to which file types *st_nlink* applies.

Changes are made related to support for finegrained timestamps.

NAME

fstatat, lstat, stat — get file status

SYNOPSIS

OH
```
#include <fcntl.h>
#include <sys/stat.h>

int fstatat(int fd, const char *restrict path,
    struct stat *restrict buf, int flag);
int lstat(const char *restrict path, struct stat *restrict buf);
int stat(const char *restrict path, struct stat *restrict buf);
```

DESCRIPTION

The *stat*( ) function shall obtain information about the named file and write it to the area pointed to by the *buf* argument. The *path* argument points to a pathname naming a file. Read, write, or execute permission of the named file is not required. An implementation that provides additional or alternate file access control mechanisms may, under implementation-defined conditions, cause *stat*( ) to fail. In particular, the system may deny the existence of the file specified by *path*.

If the named file is a symbolic link, the *stat*( ) function shall continue pathname resolution using the contents of the symbolic link, and shall return information pertaining to the resulting file if the file exists.

The *buf* argument is a pointer to a **stat** structure, as defined in the **<sys/stat.h>** header, into which information is placed concerning the file.

The *stat*( ) function shall update any time-related fields (as described in XBD Section 4.12, on page 98), before writing into the **stat** structure.

SHM If the named file is a shared memory object, the implementation shall update in the **stat** structure pointed to by the *buf* argument the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be valid. The implementation may update other fields and flags.

TYM If the named file is a typed memory object, the implementation shall update in the **stat** structure pointed to by the *buf* argument the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be valid. The implementation may update other fields and flags.

For all other file types defined in this volume of POSIX.1-2024, the structure members *st_mode*, *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atim*, *st_ctim*, and *st_mtim* shall have meaningful values and the value of the member *st_nlink* shall be set to the number of hard links to the file.

The *lstat*( ) function shall be equivalent to *stat*( ), except when *path* refers to a symbolic link. In that case *lstat*( ) shall return information about the link, while *stat*( ) shall return information about the file the link references.

For symbolic links, the *st_mode* member shall contain meaningful information when used with the file type macros. The file mode bits in *st_mode* are unspecified. The structure members *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atim*, *st_ctim*, and *st_mtim* shall have meaningful values and the value of the *st_nlink* member shall be set to the number of hard links to the symbolic link. The value of the *st_size* member shall be set to the length of the pathname contained in the symbolic link not including any terminating null byte.

The *fstatat*( ) function shall be equivalent to the *stat*( ) or *lstat*( ) function, depending on the value of *flag* (see below), except in the case where *path* specifies a relative path. In this case the status shall be retrieved from a file relative to the directory associated with the file descriptor *fd* instead

of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in **<fcntl.h>**:

AT_SYMLINK_NOFOLLOW
If *path* names a symbolic link, the status of the symbolic link is returned.

If *fstatat*( ) is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used and the behavior shall be identical to a call to *stat*( ) or *lstat*( ) respectively, depending on whether or not the AT_SYMLINK_NOFOLLOW bit is set in *flag*.

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return –1 and set *errno* to indicate the error.

ERRORS

These functions shall fail if:

[EACCES] Search permission is denied for a component of the path prefix.

[EIO] An error occurred while reading from the file system.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of a component of a pathname is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

[EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file serial number cannot be represented correctly in the structure pointed to by *buf*.

The *fstatat*( ) function shall fail if:

[EACCES] The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

[EBADF] The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

[ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

These functions may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

[EOVERFLOW] A value to be stored would overflow one of the members of the **stat** structure.

The *fstatat*( ) function may fail if:

[EINVAL] The value of the *flag* argument is not valid.

EXAMPLES

Obtaining File Status Information

The following example shows how to obtain file status information for a file named **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

struct stat buffer;
int         status;
...
status = stat("/home/cnd/mod1", &buffer);
```

Getting Directory Information

The following example fragment gets status information for each entry in a directory. The call to the *stat*( ) function stores file information in the **stat** structure pointed to by *statbuf*. The lines that follow the *stat*( ) call format the fields in the **stat** structure for presentation to the user of the program.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>
#include <locale.h>
#include <langinfo.h>
#include <stdio.h>
#include <stdint.h>

struct dirent  *dp;
struct stat     statbuf;
struct passwd  *pwd;
struct group   *grp;
struct tm      *tm;
char            datestring[256];
...
/* Loop through directory entries. */
while ((dp = readdir(dir)) != NULL) {

    /* Get entry's information. */
    if (stat(dp->d_name, &statbuf) == -1)
```



```
            continue;

        /* Print out type, permissions, and number of links. */
        printf("%10.10s", mode_string(statbuf.st_mode));
        printf(" %4ju", (uintmax_t)statbuf.st_nlink);

        /* Print out owner's name if it is found using getpwuid(). */
        if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
            printf(" %-8.8s", pwd->pw_name);
        else
            printf(" %-8ju", (uintmax_t)statbuf.st_uid);

        /* Print out group name if it is found using getgrgid(). */
        if ((grp = getgrgid(statbuf.st_gid)) != NULL)
            printf(" %-8.8s", grp->gr_name);
        else
            printf(" %-8ju", (uintmax_t)statbuf.st_gid);

        /* Print size of file. */
        printf(" %9jd", (intmax_t)statbuf.st_size);

        tm = localtime(&statbuf.st_mtime);

        /* Get localized date string. */
        strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);

        printf(" %s %s\n", datestring, dp->d_name);
    }
```

Obtaining Symbolic Link Status Information

The following example shows how to obtain status information for a symbolic link named **/modules/pass1**. The structure variable *buffer* is defined for the **stat** structure. If the *path* argument specified the pathname for the file pointed to by the symbolic link (**/home/cnd/mod1**), the results of calling the function would be the same as those returned by a call to the *stat*( ) function.

```
#include <sys/stat.h>

struct stat buffer;
int status;
...
status = lstat("/modules/pass1", &buffer);
```

APPLICATION USAGE

None.

RATIONALE

The intent of the paragraph describing ``additional or alternate file access control mechanisms'' is to allow a secure implementation where a process with a label that does not dominate the file's label cannot perform a *stat*( ) function. This is not related to read permission; a process with a label that dominates the file's label does not need read permission. An implementation that supports write-up operations could fail *fstat*( ) function calls even though it has a valid file descriptor open for writing.

The purpose of the *fstatat*( ) function is to obtain the status of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *stat*( ), resulting in unspecified behavior. By opening a

file descriptor for the target directory and using the *fstatat*( ) function it can be guaranteed that the file for which status is returned is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

access( ), *chmod*( ), *fdopendir*( ), *fstat*( ), *mknod*( ), *readlink*( ), *symlink*( )

XBD Section 4.12 (on page 98), **<fcntl.h>**, **<sys/stat.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Large File Summit extensions are added.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [EIO] mandatory error condition is added.
- The [ELOOP] mandatory error condition is added.
- The [EOVERFLOW] mandatory error condition is added. This change is to support large files.
- The [ENAMETOOLONG] and the second [EOVERFLOW] optional error conditions are added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Details are added regarding the treatment of symbolic links.
- The [ELOOP] optional error condition is added.

The normative text is updated to avoid use of the term ``must'' for application requirements.

The **restrict** keyword is added to the *stat*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

XSH-SD5-ERN-161 is applied, updating the DESCRIPTION to clarify to which file types *st_nlink* applies.

The *fstatat*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

The *lstat*( ) function is now required to return meaningful data for symbolic links in all **stat** structure fields, except for the permission bits of *st_mode*.

Changes are made to allow a directory to be opened for searching.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0198 [461], XSH/TC1-2008/0199 [324], XSH/TC1-2008/0200 [278], XSH/TC1-2008/0201 [278], and XSH/TC1-2008/0202 [291] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0136 [591], XSH/TC2-2008/0137 [817], XSH/TC2-2008/0138 [817], and XSH/TC2-2008/0139 [889] are applied.

Issue 8

Austin Group Defect 1380 is applied, changing text using the term ``link'' in line with its updated definition.

Austin Group Defect 1409 is applied, changing the EXAMPLES section.

NAME

fstatvfs, statvfs — get file system information

SYNOPSIS

```
#include <sys/statvfs.h>

int fstatvfs(int fildes, struct statvfs *buf);
int statvfs(const char *restrict path, struct statvfs *restrict buf);
```

DESCRIPTION

The *fstatvfs*( ) function shall obtain information about the file system containing the file referenced by *fildes*.

The *statvfs*( ) function shall obtain information about the file system containing the file named by *path*.

For both functions, the *buf* argument is a pointer to a **statvfs** structure that shall be filled. Read, write, or execute permission of the named file is not required.

The following flags can be returned in the *f_flag* member:

ST_RDONLY Read-only file system.

ST_NOSUID Setuid/setgid bits ignored by *exec*.

It is unspecified whether all members of the **statvfs** structure have meaningful values on all file systems.

RETURN VALUE

Upon successful completion, *statvfs*( ) shall return 0. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *fstatvfs*( ) and *statvfs*( ) functions shall fail if:

[EIO] An I/O error occurred while reading the file system.

[EINTR] A signal was caught during execution of the function.

[EOVERFLOW] One of the values to be returned cannot be represented correctly in the structure pointed to by *buf*.

The *fstatvfs*( ) function shall fail if:

[EBADF] The *fildes* argument is not an open file descriptor.

The *statvfs*( ) function shall fail if:

[EACCES] Search permission is denied on a component of the path prefix.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG] The length of a component of a pathname is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

The *statvfs*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

EXAMPLES

Obtaining File System Information Using fstatvfs( )

The following example shows how to obtain file system information for the file system upon which the file named **/home/cnd/mod1** resides, using the *fstatvfs*( ) function. The **/home/cnd/mod1** file is opened with read/write privileges and the open file descriptor is passed to the *fstatvfs*( ) function.

```
#include <sys/statvfs.h>
#include <fcntl.h>

struct statvfs buffer;
int            status;
...
fildes = open("/home/cnd/mod1", O_RDWR);
status  = fstatvfs(fildes, &buffer);
```

Obtaining File System Information Using statvfs( )

The following example shows how to obtain file system information for the file system upon which the file named **/home/cnd/mod1** resides, using the *statvfs*( ) function.

```
#include <sys/statvfs.h>

struct statvfs buffer;
int            status;
...
status = statvfs("/home/cnd/mod1", &buffer);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *chown*( ), *creat*( ), *dup*( ), *exec*, *fcntl*( ), *futimens*( ), *link*( ), *mknod*( ), *open*( ), *pipe*( ), *read*( ), *time*( ), *unlink*( ), *write*( )

XBD **<sys/statvfs.h>**

36452 **CHANGE HISTORY**

36453 First released in Issue 4, Version 2.

36454 **Issue 5**

36455 Moved from X/OPEN UNIX extension to BASE.

36456 Large File Summit extensions are added.

36457 **Issue 6**

36458 The normative text is updated to avoid use of the term ``must'' for application requirements.

36459 The **restrict** keyword is added to the *statvfs*( ) prototype for alignment with the
36460 ISO/IEC 9899: 1999 standard.

36461 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
36462 [ELOOP] error condition is added.

36463 **Issue 7**

36464 Austin Group Interpretation 1003.1-2001 #143 is applied.

36465 SD5-XSH-ERN-68 is applied, correcting the EXAMPLES section.

36466 The *fstatvfs*( ) and *statvfs*( ) functions are moved from the XSI option to the Base.

36467 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
36468 pathname exists but is not a directory or a symbolic link to a directory.

36469 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0203 [324] is applied.

NAME

fsync — synchronize changes to a file

SYNOPSIS

FSC
```
#include <unistd.h>

int fsync(int fildes);
```

DESCRIPTION

The *fsync*( ) function shall request that all data for the open file descriptor named by *fildes* is to be transferred to the storage device associated with the file described by *fildes*. The nature of the transfer is implementation-defined. The *fsync*( ) function shall not return until the system has completed that action or until an error is detected.

SIO If _POSIX_SYNCHRONIZED_IO is defined, the *fsync*( ) function shall force all currently queued I/O operations associated with the file indicated by file descriptor *fildes* to the synchronized I/O completion state. All I/O operations shall be completed as defined for synchronized I/O file integrity completion.

RETURN VALUE

Upon successful completion, *fsync*( ) shall return 0. Otherwise, –1 shall be returned and *errno* set to indicate the error. If the *fsync*( ) function fails, outstanding I/O operations are not guaranteed to have been completed.

ERRORS

The *fsync*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid descriptor.

[EINTR] The *fsync*( ) function was interrupted by a signal.

[EINVAL] The *fildes* argument does not refer to a file on which this operation is possible.

[EIO] An I/O error occurred while reading from or writing to the file system.

In the event that any of the queued I/O operations fail, *fsync*( ) shall return the error conditions defined for *read*( ) and *write*( ).

EXAMPLES

None.

APPLICATION USAGE

The *fsync*( ) function should be used by programs which require modifications to a file to be completed before continuing; for example, a program which contains a simple transaction facility might use it to ensure that all modifications to a file or files caused by a transaction are recorded.

An application that modifies a directory, for example, by creating a file in the directory, can invoke *fsync*( ) on the directory to ensure that the directory's entries and file attributes are synchronized. For most applications, synchronizing the directory's entries should not be necessary (see XBD Section 4.11, on page 98).

RATIONALE

The *fsync*( ) function is intended to force a physical write of data from the buffer cache, and to assure that after a system crash or other failure that all data up to the time of the *fsync*( ) call is recorded on the disk. Since the concepts of ``buffer cache'', ``system crash'', ``physical write'', and ``non-volatile storage'' are not defined here, the wording has to be more abstract.

If _POSIX_SYNCHRONIZED_IO is not defined, the wording relies heavily on the conformance

document to tell the user what can be expected from the system. It is explicitly intended that a null implementation is permitted. This could be valid in the case where the system cannot assure non-volatile storage under any circumstances or when the system is highly fault-tolerant and the functionality is not required. In the middle ground between these extremes, *fsync*( ) might or might not actually cause data to be written where it is safe from a power failure. The conformance document should identify at least that one configuration exists (and how to obtain that configuration) where this can be assured for at least some files that the user can select to use for critical data. It is not intended that an exhaustive list is required, but rather sufficient information is provided so that if critical data needs to be saved, the user can determine how the system is to be configured to allow the data to be written to non-volatile storage.

It is reasonable to assert that the key aspects of *fsync*( ) are unreasonable to test in a test suite. That does not make the function any less valuable, just more difficult to test. A formal conformance test should probably force a system crash (power shutdown) during the test for this condition, but it needs to be done in such a way that automated testing does not require this to be done except when a formal record of the results is being made. It would also not be unreasonable to omit testing for *fsync*( ), allowing it to be treated as a quality-of-implementation issue.

FUTURE DIRECTIONS

None.

SEE ALSO

sync( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 3.

Issue 5

Aligned with *fsync*( ) in the POSIX Realtime Extension. Specifically, the DESCRIPTION and RETURN VALUE sections are much expanded, and the ERRORS section is updated to indicate that *fsync*( ) can return the error conditions defined for *read*( ) and *write*( ).

Issue 6

This function is marked as part of the File Synchronization option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EINVAL] and [EIO] mandatory error conditions are added.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/44 is applied, applying an editorial rewording of the DESCRIPTION. No change in meaning is intended.

Issue 8

Austin Group Defect 672 is applied, changing the APPLICATION USAGE section.

NAME

ftell, ftello — return a file offset in a stream

SYNOPSIS

```
#include <stdio.h>

long ftell(FILE *stream);
CX off_t ftello(FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *ftell*( ) function shall obtain the current value of the file-position indicator for the stream pointed to by *stream*.

The *ftell*( ) function shall not change the setting of *errno* if successful.

CX The *ftello*( ) function shall be equivalent to *ftell*( ), except that the return value is of type **off_t** and the *ftello*( ) function may change the setting of *errno* if successful.

RETURN VALUE

CX Upon successful completion, *ftell*( ) and *ftello*( ) shall return the current value of the file-position CX indicator for the stream measured in bytes from the beginning of the file, except in the case of streams opened with *open_wmemstream*( ) for which the position shall be measured in wide characters.

Otherwise, *ftell*( ) and *ftello*( ) shall return –1, and set *errno* to indicate the error.

ERRORS

CX The *ftell*( ) and *ftello*( ) functions shall fail if:

CX [EBADF] The file descriptor underlying *stream* is not an open file descriptor.

CX [EOVERFLOW] For *ftell*( ), the current file offset cannot be represented correctly in an object of type **long**.

CX [EOVERFLOW] For *ftello*( ), the current file offset cannot be represented correctly in an object of type **off_t**.

CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

For all streams other than those opened by *open_wmemstream*( ), *ftell*( ) and *fseek*( ) operate on byte offsets. The behavior with *open_wmemstream*( ) streams is intentionally different—*ftell*( ) and *fseek*( ) operate on wide character offsets. This is because those streams are unique in that the backing storage is not a multibyte representation but a wide character array, and it is useful to be able to use the output of *ftell*( ) to index into that array.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fgetpos*( ), *fopen*( ), *fseek*( ), *lseek*( ), *open_memstream*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The *ftello*( ) function is added.
- The [EOVERFLOW] error conditions are added.

An additional [ESPIPE] error condition is added for sockets.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0204 [105], XSH/TC1-2008/0205 [421], XSH/TC1-2008/0206 [122], XSH/TC1-2008/0207 [122], and XSH/TC1-2008/0208 [14] are applied.

Issue 8

Austin Group Defect 1027 is applied, specifying that for streams opened with *open_wmemstream*( ) the position is measured in wide characters, not bytes.

NAME

ftok — generate an IPC key

SYNOPSIS

XSI
```
#include <sys/ipc.h>

key_t ftok(const char *path, int id);
```

DESCRIPTION

The *ftok*( ) function shall return a key based on *path* and *id* that is usable in subsequent calls to *msgget*( ), *semget*( ), and *shmget*( ). The application shall ensure that the *path* argument is the pathname of an existing file that the process is able to *stat*( ), with the exception that if *stat*( ) would fail with [EOVERFLOW] due to file size, *ftok*( ) shall still succeed.

The *ftok*( ) function shall return the same key value for all paths that name the same file, when called with the same *id* value, and should return different key values when called with different *id* values or with paths that name different files existing on the same file system at the same time. It is unspecified whether *ftok*( ) shall return the same key value when called again after the file named by *path* is removed and recreated with the same name.

Only the low-order 8-bits of *id* are significant. The behavior of *ftok*( ) is unspecified if these bits are 0.

RETURN VALUE

Upon successful completion, *ftok*( ) shall return a key. Otherwise, *ftok*( ) shall return (**key_t**)–1 and set *errno* to indicate the error.

ERRORS

The *ftok*( ) function shall fail if:

[EACCES] Search permission is denied for a component of the path prefix.

[EIO] An error occurred while reading from the file system.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of a component of a pathname is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

The *ftok*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

EXAMPLES

Getting an IPC Key

The following example gets a key based on the pathname **/tmp** and the ID value *a*. It also assigns the value of the resulting key to the *semkey* variable so that it will be available to a later call to *semget*( ), *msgget*( ), or *shmget*( ).

```
#include <sys/ipc.h>
...
key_t semkey;

if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
    perror("IPC error: ftok"); exit(1);
}
```

APPLICATION USAGE

For maximum portability, *id* should be a single-byte character.

Applications should not assume that the resulting key value is unique.

RATIONALE

None.

FUTURE DIRECTIONS

Future versions of this standard may add new interfaces to provide unique keys.

SEE ALSO

msgget( ), *semget*( ), *shmget*( )

XBD **<sys/ipc.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0209 [343], XSH/TC1-2008/0210 [366], XSH/TC1-2008/0211 [343], XSH/TC1-2008/0212 [324], XSH/TC1-2008/0213 [366], XSH/TC1-2008/0214 [366], XSH/TC1-2008/0215 [366], and XSH/TC1-2008/0216 [366] are applied.

NAME

ftruncate — truncate a file to a specified length

SYNOPSIS

```
#include <unistd.h>

int ftruncate(int fildes, off_t length);
```

DESCRIPTION

If *fildes* is not a valid file descriptor open for writing, the *ftruncate*( ) function shall fail.

If *fildes* refers to a regular file, the *ftruncate*( ) function shall cause the size of the file to be truncated to *length*. If the size of the file previously exceeded *length*, the extra data shall no longer be available to reads on the file. If the file previously was smaller than this size, *ftruncate*( ) shall increase the size of the file. If the file size is increased, the extended area shall appear as if it were zero-filled. The value of the seek pointer shall not be modified by a call to *ftruncate*( ).

Upon successful completion, if *fildes* refers to a regular file, *ftruncate*( ) shall mark for update the last data modification and last file status change timestamps of the file and the S_ISUID and S_ISGID bits of the file mode may be cleared. If the *ftruncate*( ) function is unsuccessful, the file is unaffected.

If the request would cause the file size to exceed the soft file size limit for the process, the request shall fail XSI and the implementation shall generate a SIGXFSZ signal for the thread.

If *fildes* is a file descriptor open for writing and refers to a file that is neither a regular file nor a shared memory object, the result is unspecified.

SHM If *fildes* refers to a shared memory object, *ftruncate*( ) shall set the size of the shared memory object to *length*.

SHM If the effect of *ftruncate*( ) is to decrease the size of a memory mapped file or a shared memory object and whole pages beyond the new end were previously mapped, then the whole pages beyond the new end shall be discarded.

References to discarded pages shall result in the generation of a SIGBUS signal.

If the effect of *ftruncate*( ) is to increase the size of a memory object, it is unspecified whether the contents of any mapped pages between the old end-of-file and the new are flushed to the underlying object.

RETURN VALUE

Upon successful completion, *ftruncate*( ) shall return 0; otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *ftruncate*( ) function shall fail if:

[EBADF] or [EINVAL]
: The *fildes* argument is not a file descriptor open for writing.

[EFBIG] or [EINVAL]
: The *length* argument is greater than the maximum file size.

XSI [EFBIG]
: The *length* argument exceeds the file size limit of the process. A SIGXFSZ signal shall also be generated for the thread.

[EFBIG]
: The file is a regular file and *length* is greater than the offset maximum established in the open file description associated with *fildes*.

[EINTR] A signal was caught during execution.

[EINVAL] The *length* argument is less than 0 or the *fildes* argument refers to a file on which this operation is not possible (for example, a pipe, FIFO or socket).

[EIO] An I/O error occurred while reading from or writing to a file system.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

open( ), *truncate*( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE and aligned with *ftruncate*( ) in the POSIX Realtime Extension. Specifically, the DESCRIPTION is extensively reworded and [EROFS] is added to the list of mandatory errors that can be returned by *ftruncate*( ).

Large File Summit extensions are added.

Issue 6

The *truncate*( ) function is split out into a separate reference page.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is changed to indicate that if the file size is changed, and if the file is a regular file, the S_ISUID and S_ISGID bits in the file mode may be cleared.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The DESCRIPTION text is updated.

XSI-conformant systems are required to increase the size of the file if the file was previously smaller than the size requested.

Issue 7

Austin Group Interpretation 1003.1-2001 #056 is applied, revising the ERRORS section (although the [EINVAL] ``may fail'' error was subsequently removed during review of the XSI option).

Functionality relating to the Memory Protection and Memory Mapped Files options is moved to the Base.

The DESCRIPTION is updated so that a call to *ftruncate*( ) when the file is smaller than the size requested will increase the size of the file. Previously, non-XSI-conforming implementations were allowed to increase the size of the file or fail.

Changes are made related to support for finegrained timestamps.

Issue 8

Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

Austin Group Defect 1169 is applied, removing a redundant statement that *ftruncate*( ) fails if *fildes* refers to a directory.

Austin Group Defect 1381 is applied, adding a second condition to the [EINVAL] error and rearranging the ERRORS section into alphabetical order.

Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error relating to the file size limit for the process.

NAME

ftrylockfile — stdio locking functions

SYNOPSIS

CX
```
#include <stdio.h>

int ftrylockfile(FILE *file);
```

DESCRIPTION

Refer to *flockfile( )*.

36789 **NAME**

36790 funlockfile — stdio locking functions

36791 **SYNOPSIS**

36792 CX
```
#include <stdio.h>

void funlockfile(FILE *file);
```
36793

36794 **DESCRIPTION**

36795 Refer to *flockfile( )*.

NAME

futimens, utimensat, utimes — set file access and modification times

SYNOPSIS

```
#include <sys/stat.h>

int futimens(int fd, const struct timespec times[2]);
```

OH
```
#include <fcntl.h>

int utimensat(int fd, const char *path, const struct timespec times[2],
    int flag);
```

XSI
```
#include <sys/time.h>

int utimes(const char *path, const struct timeval times[2]);
```

DESCRIPTION

The *futimens*( ) and *utimensat*( ) functions shall set the access and modification times of a file to the values of the *times* argument. The *futimens*( ) function changes the times of the file associated with the file descriptor *fd*. The *utimensat*( ) function changes the times of the file pointed to by the *path* argument, relative to the directory associated with the file descriptor *fd*. Both functions allow time specifications accurate to the nanosecond.

For *futimens*( ) and *utimensat*( ), the *times* argument is an array of two **timespec** structures. The first array member represents the date and time of last access, and the second member represents the date and time of last modification. The times in the **timespec** structure are measured in seconds and nanoseconds since the Epoch. The file's relevant timestamp shall be set to the greatest value supported by the file system that is not greater than the specified time.

If the *tv_nsec* field of a **timespec** structure has the special value UTIME_NOW, the file's relevant timestamp shall be set to the greatest value supported by the file system that is not greater than the current time. If the *tv_nsec* field has the special value UTIME_OMIT, the file's relevant timestamp shall not be changed. In either case, the *tv_sec* field shall be ignored.

If the *times* argument is a null pointer, both the access and modification timestamps shall be set to the greatest value supported by the file system that is not greater than the current time. If *utimensat*( ) is passed a relative path in the *path* argument, the file to be used shall be relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

If *utimensat*( ) is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used.

Only a process with the effective user ID equal to the user ID of the file, or with write access to the file, or with appropriate privileges may use *futimens*( ) or *utimensat*( ) with a null pointer as the *times* argument or with both *tv_nsec* fields set to the special value UTIME_NOW. Only a process with the effective user ID equal to the user ID of the file or with appropriate privileges may use *futimens*( ) or *utimensat*( ) with a non-null *times* argument that does not have both *tv_nsec* fields set to UTIME_NOW and does not have both *tv_nsec* fields set to UTIME_OMIT. If both *tv_nsec* fields are set to UTIME_OMIT, no ownership or permissions check shall be performed for the file, but other error conditions may still be detected (including [EACCES] errors related to the path prefix).

Values for the *flag* argument of *utimensat*( ) are constructed by a bitwise-inclusive OR of flags

from the following list, defined in **<fcntl.h>**:

AT_SYMLINK_NOFOLLOW
: If *path* names a symbolic link, then the access and modification times of the symbolic link are changed.

Upon successful completion, *futimens*( ) and *utimensat*( ) shall mark the last file status change timestamp for update, with the exception that if both *tv_nsec* fields are set to UTIME_OMIT, the file status change timestamp need not be marked for update.

The *utimes*( ) function shall be equivalent to the *utimensat*( ) function with the special value AT_FDCWD as the *fd* argument and the *flag* argument set to zero, except that the *times* argument is a **timeval** structure rather than a **timespec** structure, and accuracy is only to the microsecond, not nanosecond, and rounding towards the nearest second may occur.

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return –1 and set *errno* to indicate the error. If –1 is returned, the file times shall not be affected.

ERRORS

The *utimes*( ) function shall fail, the *futimens*( ) and *utimensat*( ) functions shall fail in the case that the *times* argument does not have both *tv_nsec* fields set to UTIME_OMIT, and the *futimens*( ) and *utimensat*( ) functions may fail in the case that the *times* argument has both *tv_nsec* fields set to UTIME_OMIT, if:

[EACCES] The *times* argument is a null pointer, or both *tv_nsec* values are UTIME_NOW, and the effective user ID of the process does not match the owner of the file and write access is denied.

[EINVAL] Either of the *times* argument structures specified a *tv_nsec* value that was neither UTIME_NOW nor UTIME_OMIT, and was a value less than zero or greater than or equal to 1 000 million.

[EINVAL] A new file timestamp would be a value whose *tv_sec* component is not a value supported by the file system.

[EPERM] The *times* argument is not a null pointer, does not have both *tv_nsec* fields set to UTIME_NOW, does not have both *tv_nsec* fields set to UTIME_OMIT, the calling process' effective user ID does not match the owner of the file, and the calling process does not have appropriate privileges.

[EROFS] The file system containing the file is read-only.

The *futimens*( ) function shall fail in the case that the *times* argument does not have both *tv_nsec* fields set to UTIME_OMIT, and may fail in the case that the *times* argument has both *tv_nsec* fields set to UTIME_OMIT, if:

[EBADF] The *fd* argument is not a valid file descriptor.

The *utimensat*( ) function shall fail in the case that the *times* argument does not have both *tv_nsec* fields set to UTIME_OMIT, and may fail in the case that the *times* argument has both *tv_nsec* fields set to UTIME_OMIT, if:

[EACCES] The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

[EBADF] The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

[ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

The *utimes*( ) function shall fail, the *utimensat*( ) function shall fail in the case that the *times* argument does not have both *tv_nsec* fields set to UTIME_OMIT, and the *utimensat*( ) function may fail in the case that the *times* argument has both *tv_nsec* fields set to UTIME_OMIT, if:

[EACCES] Search permission is denied by a component of the path prefix.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of a component of a pathname is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

The *utimensat*( ) and *utimes*( ) functions may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The *utimensat*( ) function may fail if:

[EINVAL] The value of the *flag* argument is not valid.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The purpose of the *utimensat*( ) function is to set the access and modification time of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *utimes*( ), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *utimensat*( ) function it can be guaranteed that the changed file is located relative to the desired directory.

The standard developers considered including a special case for the permissions required by *utimensat*( ) when one *tv_nsec* field is UTIME_NOW and the other is UTIME_OMIT. One possibility would be to include this case in with the cases where *times* is a null pointer or both fields are UTIME_NOW, where the call is allowed if the process has write permission for the file. However, associating write permission with an update to just the last data access timestamp (which is normally updated by *read*( )) did not seem appropriate. The other possibility would be to specify that this one case is allowed if the process has read permission, but this was felt to be

too great a departure from the *utime*( ) and *utimes*( ) functions on which *utimensat*( ) is based. If an application needs to set the last data access timestamp to the current time for a file on which it has read permission but is not the owner, it can do so by opening the file, reading one or more bytes (or reading a directory entry, if the file is a directory), and then closing it.

FUTURE DIRECTIONS

None.

SEE ALSO

read( )

XBD **<fcntl.h>**, **<sys/stat.h>**, **<sys/time.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

This function is marked LEGACY.

The normative text is updated to avoid use of the term ``must'' for application requirements.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

The LEGACY marking is removed.

The *utimensat*( ) function (renamed from *futimesat*( )) is added from The Open Group Technical Standard, 2006, Extended API Set Part 2, and changed to allow modifying a symbolic link by adding a *flag* argument.

The *futimens*( ) function is added.

Changes are made related to support for finegrained timestamps.

Changes are made to allow a directory to be opened for searching.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0220 [63,428], XSH/TC1-2008/0221 [278], XSH/TC1-2008/0222 [324], XSH/TC1-2008/0223 [306], and XSH/TC1-2008/0224 [278] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0140 [591], XSH/TC2-2008/0141 [817], XSH/TC2-2008/0142 [485], and XSH/TC2-2008/0143 [817] are applied.

Issue 8

Austin Group Defect 1280 is applied, changing the ERRORS section.

NAME

fwide — set stream orientation

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int fwide(FILE *stream, int mode);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fwide*( ) function shall determine the orientation of the stream pointed to by *stream*. If *mode* is greater than zero, the function first attempts to make the stream wide-oriented. If *mode* is less than zero, the function first attempts to make the stream byte-oriented. Otherwise, *mode* is zero and the function does not alter the orientation of the stream.

If the orientation of the stream has already been determined, *fwide*( ) shall not change it.

CX The *fwide*( ) function shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *fwide*( ), then check *errno*, and if it is non-zero, assume an error has occurred.

RETURN VALUE

The *fwide*( ) function shall return a value greater than zero if, after the call, the stream has wide-orientation, a value less than zero if the stream has byte-orientation, or zero if the stream has no orientation.

ERRORS

The *fwide*( ) function may fail if:

CX [EBADF] The *stream* argument is not a valid stream.

EXAMPLES

None.

APPLICATION USAGE

A call to *fwide*( ) with *mode* set to zero can be used to determine the current orientation of a stream.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

Extensions beyond the ISO C standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0225 [272] is applied.

NAME

fwprintf, swprintf, wprintf — print formatted wide-character output

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int fwprintf(FILE *restrict stream, const wchar_t *restrict format, ...);
int swprintf(wchar_t *restrict ws, size_t n,
    const wchar_t *restrict format, ...);
int wprintf(const wchar_t *restrict format, ...);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fwprintf*( ) function shall place output on the named output *stream*. The *wprintf*( ) function shall place output on the standard output stream *stdout*. The *swprintf*( ) function shall place output followed by the null wide character in consecutive wide characters starting at **ws*; no more than *n* wide characters shall be written, including a terminating null wide character, which is always added (unless *n* is zero).

Each of these functions shall convert, format, and print its arguments under control of the *format* wide-character string. The *format* is composed of zero or more directives: *ordinary wide-characters*, which are simply copied to the output stream, and *conversion specifications*, each of which results in the fetching of zero or more arguments. The results are undefined if there are insufficient arguments for the *format*. If the *format* is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than to the next unused argument. In this case, the conversion specifier wide character `%` (see below) is replaced by the sequence `"%n$"`, where *n* is a decimal integer in the range [1,{NL_ARGMAX}], giving the position of the argument in the argument list. This feature provides for the definition of *format* wide-character strings that select arguments in an order appropriate to specific languages (see the EXAMPLES section).

The *format* can contain either numbered argument conversion specifications (that is, those introduced by `"%n$"` and optionally containing the `"*m$"` forms of field width and precision), or unnumbered argument conversion specifications (that is, those introduced by the `%` character and optionally containing the `*` form of field width and precision), but not both. The only exception to this is that `%%` can be mixed with the `"%n$"` form. The results of mixing numbered and unnumbered argument specifications in a *format* wide-character string are undefined. When numbered argument specifications are used, specifying the *N*th argument requires that all the leading arguments, from the first to the (*N*–1)th, are specified in the *format* wide-character string.

In *format* wide-character strings containing the `"%n$"` form of conversion specification, numbered arguments in the argument list can be referenced from the *format* wide-character string as many times as required.

In *format* wide-character strings containing the `%` form of conversion specification, each argument in the argument list shall be used exactly once. It is unspecified whether an encoding error occurs if the format string contains **wchar_t** values that do not correspond to members of the character set of the current locale and the specified semantics do not require that value to be processed by *wcrtomb*( ).

37053 CX All forms of the *fwprintf*( ) function allow for the insertion of a locale-dependent radix character
37054 in the output string, output as a wide-character value. The radix character is defined in the
37055 current locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix
37056 character is not defined, the radix character shall default to a <period> (`'.'`).

37057 CX Each conversion specification is introduced by the `'%'` wide character or by the wide-character
37058 sequence `"%`*n*`$"`, after which the following appear in sequence:

37059 • Zero or more *flags* (in any order), which modify the meaning of the conversion
37060 specification.

37061 • An optional minimum *field width*. If the converted value has fewer wide characters than
37062 the field width, it shall be padded with <space> characters by default on the left; it shall be
37063 padded on the right, if the left-adjustment flag (`'-'`), described below, is given to the field
37064 CX width. The field width takes the form of an <asterisk> (`'*'`), or in conversion
37065 specifications introduced by `"%`*n*`$"` the `"*`*m*`$"` string, described below, or a decimal
37066 integer.

37067 • An optional *precision* that gives the minimum number of digits to appear for the `d`, `i`, `o`, `u`,
37068 `x`, and `X` conversion specifiers; the number of digits to appear after the radix character for
37069 the `a`, `A`, `e`, `E`, `f`, and `F` conversion specifiers; the maximum number of significant digits for
37070 the `g` and `G` conversion specifiers; or the maximum number of wide characters to be
37071 printed from a string in the `s` conversion specifiers. The precision takes the form of a
37072 CX <period> (`'.'`) followed either by an <asterisk> (`'*'`), or in conversion specifications
37073 introduced by `"%`*n*`$"` the `"*`*m*`$"` string, described below, or an optional decimal digit
37074 string, where a null digit string is treated as 0. If a precision appears with any other
37075 conversion wide character, the behavior is undefined.

37076 • An optional length modifier that specifies the size of the argument.

37077 • A *conversion specifier* wide character that indicates the type of conversion to be applied.

37078 A field width, or precision, or both, may be indicated by an <asterisk> (`'*'`). In this case an
37079 argument of type **int** supplies the field width or precision. Applications shall ensure that
37080 arguments specifying field width, or precision, or both appear in that order before the argument,
37081 if any, to be converted. A negative field width is taken as a `'-'` flag followed by a positive field
37082 CX width. A negative precision is taken as if the precision were omitted. In *format* wide-character
37083 strings containing conversion specifications introduced by `"%`*n*`$"`, in addition to being indicated
37084 by the decimal digit string, a field width may be indicated by the sequence `"*`*m*`$"` and precision
37085 by the sequence `".*`*m*`$"`, where *m* is a decimal integer in the range [1,{NL_ARGMAX}] giving
37086 the position in the argument list (after the *format* argument) of an integer argument containing
37087 the field width or precision, for example:

37088 ```
wprintf(L"%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

37089 The flag wide characters and their meanings are:

37090 CX `'` (The <apostrophe>.) The integer portion of the result of a decimal conversion (`%i`, `%d`,
37091 `%u`, `%f`, `%F`, `%g`, or `%G`) shall be formatted with thousands' grouping wide characters. For
37092 other conversions, the behavior is undefined. The numeric grouping wide character is
37093 used.

37094 `-` The result of the conversion shall be left-justified within the field. The conversion shall
37095 be right-justified if this flag is not specified.

37096 `+` The result of a signed conversion shall always begin with a sign (`'+'` or `'-'`). The
37097 conversion shall begin with a sign only when a negative value is converted if this flag is
37098 not specified.

  
```

<space> If the first wide character of a signed conversion is not a sign, or if a signed conversion results in no wide characters, a <space> shall be prefixed to the result. This means that if the <space> and '+' flags both appear, the <space> flag shall be ignored.

Specifies that the value is to be converted to an alternative form. For o conversion, it shall increase the precision, if and only if necessary, to force the first digit of the result to be zero (if the value and precision are both 0, a single 0 is printed). For x or X conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A, e, E, f, F, g, and G conversion specifiers, the result shall always contain a radix character, even if no digits follow it. Without this flag, a radix character appears in the result of these conversions only if a digit follows it. For g and G conversion specifiers, trailing zeros shall *not* be removed from the result as they normally are. For other conversion specifiers, the behavior is undefined.

0 For d, i, o, u, x, X, a, A, e, E, f, F, g, and G conversion specifiers, leading zeros (following any indication of sign or base) are used to pad to the field width rather than performing space padding, except when converting an infinity or NaN. If the '0' and '−' flags both appear, the '0' flag shall be ignored. For d, i, o, u, x, and X conversion specifiers, if a precision is specified, the '0' flag shall be ignored. If the '0' and <apostrophe> flags both appear, the grouping wide characters are inserted before zero padding. For other conversions, the behavior is undefined.

CX

The length modifiers and their meanings are:

hh Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **signed char** or **unsigned char** argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to **signed char** or **unsigned char** before printing); or that a following n conversion specifier applies to a pointer to a **signed char** argument.

h Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **short** or **unsigned short** argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to **short** or **unsigned short** before printing); or that a following n conversion specifier applies to a pointer to a **short** argument.

l (ell) Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long** or **unsigned long** argument; that a following n conversion specifier applies to a pointer to a **long** argument; that a following c conversion specifier applies to a **wint_t** argument; that a following s conversion specifier applies to a pointer to a **wchar_t** argument; or has no effect on a following a, A, e, E, f, F, g, or G conversion specifier.

ll (ell-ell) Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long long** or **unsigned long long** argument; or that a following n conversion specifier applies to a pointer to a **long long** argument.

j Specifies that a following d, i, o, u, x, or X conversion specifier applies to an **intmax_t** or **uintmax_t** argument; or that a following n conversion specifier applies to a pointer to an **intmax_t** argument.

z Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **size_t** or the corresponding signed integer type argument; or that a following n conversion specifier applies to a pointer to a signed integer type corresponding to a **size_t** argument.

37144 t Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **ptrdiff_t** or
37145 the corresponding **unsigned** type argument; or that a following n conversion specifier
37146 applies to a pointer to a **ptrdiff_t** argument.

37147 L Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to a **long**
37148 **double** argument.

37149 If a length modifier appears with any conversion specifier other than as specified above, the
37150 behavior is undefined.

37151 The conversion specifiers and their meanings are:

37152 d,i The **int** argument shall be converted to a signed decimal in the style "[−]*dddd*". The
37153 precision specifies the minimum number of digits to appear; if the value being
37154 converted can be represented in fewer digits, it shall be expanded with leading zeros.
37155 The default precision shall be 1. The result of converting zero with an explicit precision
37156 of zero shall be no wide characters.

37157 o The **unsigned** argument shall be converted to unsigned octal format in the style
37158 "dddd". The precision specifies the minimum number of digits to appear; if the value
37159 being converted can be represented in fewer digits, it shall be expanded with leading
37160 zeros. The default precision shall be 1. The result of converting zero with an explicit
37161 precision of zero shall be no wide characters.

37162 u The **unsigned** argument shall be converted to unsigned decimal format in the style
37163 "dddd". The precision specifies the minimum number of digits to appear; if the value
37164 being converted can be represented in fewer digits, it shall be expanded with leading
37165 zeros. The default precision shall be 1. The result of converting zero with an explicit
37166 precision of zero shall be no wide characters.

37167 x The **unsigned** argument shall be converted to unsigned hexadecimal format in the style
37168 "dddd"; the letters "abcdef" are used. The precision specifies the minimum number
37169 of digits to appear; if the value being converted can be represented in fewer digits, it
37170 shall be expanded with leading zeros. The default precision shall be 1. The result of
37171 converting zero with an explicit precision of zero shall be no wide characters.

37172 X Equivalent to the x conversion specifier, except that letters "ABCDEF" are used instead
37173 of "abcdef".

37174 f,F The **double** argument shall be converted to decimal notation in the style
37175 "[−]*ddd.ddd*", where the number of digits after the radix character shall be equal to
37176 the precision specification. If the precision is missing, it shall be taken as 6; if the
37177 precision is explicitly zero and no '#' flag is present, no radix character shall appear. If
37178 a radix character appears, at least one digit shall appear before it. The value shall be
37179 rounded in an implementation-defined manner to the appropriate number of digits.

37180 A **double** argument representing an infinity shall be converted in one of the styles
37181 "[−]inf" or "[−]infinity"; which style is implementation-defined. A **double**
37182 argument representing a NaN shall be converted in one of the styles "[−]nan" or
37183 "[−]nan(*n-char-sequence*)"; which style, and the meaning of any *n-char-sequence*,
37184 is implementation-defined. The F conversion specifier produces "INF", "INFINITY",
37185 or "NAN" instead of "inf", "infinity", or "nan", respectively.

37186 e,E The **double** argument shall be converted in the style "[−]*d.ddd*e±dd", where there
37187 shall be one digit before the radix character (which is non-zero if the argument is non-
37188 zero) and the number of digits after it shall be equal to the precision; if the precision is
37189 missing, it shall be taken as 6; if the precision is zero and no '#' flag is present, no

radix character shall appear. The value shall be rounded in an implementation-defined manner to the appropriate number of digits. The E conversion wide character shall produce a number with 'E' instead of 'e' introducing the exponent. The exponent shall always contain at least two digits. If the value is zero, the exponent shall be zero.

A **double** argument representing an infinity or NaN shall be converted in the style of an f or F conversion specifier.

g, G The **double** argument representing a floating-point number shall be converted in the style f or e (or in the style F or E in the case of a G conversion specifier), depending on the value converted and the precision. Let P equal the precision if non-zero, 6 if the precision is omitted, or 1 if the precision is zero. Then, if a conversion with style E would have an exponent of *X*:

— If P>*X*≥–4, the conversion shall be with style f (or F) and precision P–(*X*+1).

— Otherwise, the conversion shall be with style e (or E) and precision P–1.

Finally, unless the '#' flag is used, any trailing zeros shall be removed from the fractional portion of the result and the decimal-point character shall be removed if there is no fractional portion remaining.

A **double** argument representing an infinity or NaN shall be converted in the style of an f or F conversion specifier.

a, A A **double** argument representing a floating-point number shall be converted in the style "[−]0x*h*.*hhhh*p±*d*", where there shall be one hexadecimal digit (which is non-zero if the argument is a normalized floating-point number and is otherwise unspecified) before the decimal-point wide character and the number of hexadecimal digits after it shall be equal to the precision; if the precision is missing and FLT_RADIX is a power of 2, then the precision shall be sufficient for an exact representation of the value; if the precision is missing and FLT_RADIX is not a power of 2, then the precision shall be sufficient to distinguish values of type **double**, except that trailing zeros may be omitted; if the precision is zero and the '#' flag is not specified, no decimal-point wide character shall appear. The letters "abcdef" are used for a conversion and the letters "ABCDEF" for A conversion. The A conversion specifier produces a number with 'X' and 'P' instead of 'x' and 'p'. The exponent shall always contain at least one digit, and only as many more digits as necessary to represent the decimal exponent of 2. If the value is zero, the exponent shall be zero.

A **double** argument representing an infinity or NaN shall be converted in the style of an f or F conversion specifier.

c If no l (ell) qualifier is present, the **int** argument shall be converted to a wide character as if by calling the *btowc*() function and the resulting wide character shall be written. Otherwise, the **wint_t** argument shall be converted to **wchar_t**, and written.

s If no l (ell) qualifier is present, the application shall ensure that the argument is a pointer to a character array containing a character sequence beginning in the initial shift state. Characters from the array shall be converted as if by repeated calls to the *mbrtowc*() function, with the conversion state described by an **mbstate_t** object initialized to zero before the first character is converted, and written up to (but not including) the terminating null wide character. If the precision is specified, no more than that many wide characters shall be written. If the precision is not specified, or is greater than the size of the array, the application shall ensure that the array contains a null wide character.

If an `l` (ell) qualifier is present, the application shall ensure that the argument is a pointer to an array of type **wchar_t**. Wide characters from the array shall be written up to (but not including) a terminating null wide character. If no precision is specified, or is greater than the size of the array, the application shall ensure that the array contains a null wide character. If a precision is specified, no more than that many wide characters shall be written.

`p` The application shall ensure that the argument is a pointer to **void**. The value of the pointer shall be converted to a sequence of printable wide characters in an implementation-defined manner.

`n` The application shall ensure that the argument is a pointer to an integer into which is written the number of wide characters written to the output so far by this call to one of the *fwprintf*( ) functions. No argument shall be converted, but one shall be consumed. If the conversion specification includes any flags, a field width, or a precision, the behavior is undefined.

XSI `C` Equivalent to `lc`.

XSI `S` Equivalent to `ls`.

`%` Write a `'%'` wide character; no argument shall be converted. The application shall ensure that the complete conversion specification is `%%`.

If a conversion specification does not match one of the above forms, the behavior is undefined.

In no case does a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field shall be expanded to contain the conversion result. Characters generated by *fwprintf*( ) and *wprintf*( ) shall be printed as if *fputwc*( ) had been called.

For `a` and `A` conversions, if FLT_RADIX is not a power of 2 and the result is not exactly representable in the given precision, the result should be one of the two adjacent numbers in hexadecimal floating style with the given precision, with the extra stipulation that the error should have a correct sign for the current rounding direction.

For `e`, `E`, `f`, `F`, `g`, and `G` conversion specifiers, if the number of significant decimal digits is at most DECIMAL_DIG, then the result should be correctly rounded. If the number of significant decimal digits is more than DECIMAL_DIG but the source value is exactly representable with DECIMAL_DIG digits, then the result should be an exact representation with trailing zeros. Otherwise, the source value is bounded by two adjacent decimal strings $L < U$, both having DECIMAL_DIG significant digits; the value of the resultant decimal string D should satisfy $L <= D <= U$, with the extra stipulation that the error should have a correct sign for the current rounding direction.

CX The last data modification and last file status change timestamps of the file shall be marked for update between the call to a successful execution of *fwprintf*( ) or *wprintf*( ) and the next successful completion of a call to *fflush*( ) or *fclose*( ) on the same stream, or a call to *exit*( ) or *abort*( ).

RETURN VALUE

Upon successful completion, these functions shall return the number of wide characters transmitted, excluding the terminating null wide character in the case of *swprintf*( ), or a negative value if an error was encountered, CX and set *errno* to indicate the error.

If *n* or more wide characters were requested to be written, *swprintf*( ) shall return a negative value, CX and set *errno* to indicate the error.

ERRORS

For the conditions under which *fwprintf*( ) and *wprintf*( ) fail and may fail, refer to *fputwc*( ).

In addition, all forms of *fwprintf*( ) shall fail if:

CX [EILSEQ] A wide-character code that does not correspond to a valid character has been detected.

CX [EOVERFLOW] The value to be returned is greater than {INT_MAX}.

In addition, *fwprintf*( ) and *wprintf*( ) may fail if:

CX [ENOMEM] Insufficient storage space is available.

The *swprintf*( ) shall fail if:

CX [EOVERFLOW] The number of wide characters requested to be written was *n* or more.

EXAMPLES

To print the language-independent date and time format, the following statement could be used:

```
wprintf(format, weekday, month, day, hour, min);
```

For American usage, *format* could be a pointer to the wide-character string:

```
L"%s, %s %d, %d:%.2d\n"
```

producing the message:

```
Sunday, July 3, 10:02
```

whereas for German usage, *format* could be a pointer to the wide-character string:

```
L"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

producing the message:

```
Sonntag, 3. Juli, 10:02
```

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that there are insufficient arguments for the format, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *btowc*( ), *fputwc*( ), *fwscanf*( ), *mbrtowc*( ), *setlocale*( )

XBD Chapter 7 (on page 127), **<inttypes.h>**, **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The Open Group Corrigendum U040/1 is applied to the RETURN VALUE section, describing the case if *n* or more wide characters are requested to be written using *swprintf*( ).

The normative text is updated to avoid use of the term ``must'' for application requirements.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The prototypes for *fwprintf*( ), *swprintf*( ), and *wprintf*( ) are updated.
- The DESCRIPTION is updated.
- The `hh`, `ll`, `j`, `t`, and `z` length modifiers are added.
- The `a`, `A`, and `F` conversion characters are added.
- XSI shading is removed from the description of character string representations of infinity and NaN floating-point values.

The DESCRIPTION is updated to use the terms ``conversion specifier'' and ``conversion specification'' consistently.

ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

Issue 7

Austin Group Interpretation 1003.1-2001 #161 is applied, updating the DESCRIPTION of the 0 flag.

Austin Group Interpretation 1003.1-2001 #170 is applied.

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #68 (SD5-XSH-ERN-70) is applied, revising the description of `g` and `G`.

Functionality relating to the `"%n$"` form of conversion specification and the <apostrophe> flag is moved from the XSI option to the Base.

The [EOVERFLOW] error is added for *swprintf*( ).

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0226 [302] and XSH/TC1-2008/0227 [14] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0144 [73], XSH/TC2-2008/0145 [894], XSH/TC2-2008/0146 [557], and XSH/TC2-2008/0147 [936] are applied.

Issue 8

Austin Group Defect 1021 is applied, changing ``output error'' to ``error'' in the RETURN VALUE section and changing the requirements for [EOVERFLOW].

Austin Group Defect 1137 is applied, clarifying the use of `"%`*n*`$"` and `"*`*m*`$"` in conversion specifications.

Austin Group Defect 1205 is applied, changing the description of the `%` conversion specifier.

Austin Group Defect 1219 is applied, changing the *swprintf*( )-specific [EOVERFLOW] error.

NAME

fwrite — binary output

SYNOPSIS

```
#include <stdio.h>

size_t fwrite(const void *restrict ptr, size_t size, size_t nitems,
    FILE *restrict stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fwrite*( ) function shall write, from the array pointed to by *ptr*, up to *nitems* elements whose size is specified by *size*, to the stream pointed to by *stream*. For each object, *size* calls shall be made to the *fputc*( ) function, taking the values (in order) from an array of **unsigned char** exactly overlaying the object. The file-position indicator for the stream (if defined) shall be advanced by the number of bytes successfully written. If an error occurs, the resulting value of the file-position indicator for the stream is unspecified.

CX The last data modification and last file status change timestamps of the file shall be marked for update between the successful execution of *fwrite*( ) and the next successful completion of a call to *fflush*( ) or *fclose*( ) on the same stream, or a call to *exit*( ) or *abort*( ).

RETURN VALUE

The *fwrite*( ) function shall return the number of elements successfully written, which shall be less than *nitems* only if a write error is encountered. If *size* or *nitems* is 0, *fwrite*( ) shall return 0 and the state of the stream remains unchanged. Otherwise, if a write error occurs, the error indicator for the stream shall be set, CX and *errno* shall be set to indicate the error.

ERRORS

Refer to *fputc*( ).

EXAMPLES

None.

APPLICATION USAGE

Because of possible differences in element length and byte ordering, files written using *fwrite*( ) are application-dependent, and possibly cannot be read using *fread*( ) by a different application or by the same application on a different processor.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *ferror*( ), *fopen*( ), *fprintf*( ), *putc*( ), *puts*( ), *write*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *fwrite*( ) prototype is updated.
- The DESCRIPTION is updated to clarify how the data is written out using *fputc*( ).

Issue 7

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0228 [14] is applied.

Issue 8

Austin Group Defect 1196 is applied, clarifying the RETURN VALUE section.

NAME

fwscanf, swscanf, wscanf — convert formatted wide-character input

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int fwscanf(FILE *restrict stream, const wchar_t *restrict format, ...);
int swscanf(const wchar_t *restrict ws,
    const wchar_t *restrict format, ...);
int wscanf(const wchar_t *restrict format, ...);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *fwscanf*( ) function shall read from the named input *stream*. The *wscanf*( ) function shall read from the standard input stream *stdin*. The *swscanf*( ) function shall read from the wide-character string *ws*. Each function reads wide characters, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control wide-character string *format* described below, and a set of *pointer* arguments indicating where the converted input should be stored. The result is undefined if there are insufficient arguments for the format. If the *format* is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than to the next unused argument. In this case, the conversion specifier wide character `%` (see below) is replaced by the sequence "`%`*n*`$`", where *n* is a decimal integer in the range [1,{NL_ARGMAX}]. This feature provides for the definition of *format* wide-character strings that select arguments in an order appropriate to specific languages. In *format* wide-character strings containing the "`%`*n*`$`" form of conversion specifications, it is unspecified whether numbered arguments in the argument list can be referenced from the *format* wide-character string more than once.

The *format* can contain either form of a conversion specification—that is, `%` or "`%`*n*`$`"— but the two forms cannot normally be mixed within a single *format* wide-character string. The only exception to this is that `%%` or `%*` can be mixed with the "`%`*n*`$`" form. When numbered argument specifications are used, specifying the *N*th argument requires that all the leading arguments, from the first to the (*N*−1)th, are pointers.

The *fwscanf*( ) function in all its forms allows for detection of a language-dependent radix character in the input string, encoded as a wide-character value. The radix character is defined in the current locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a <period> ('`.`').

The *format* is a wide-character string composed of zero or more directives. Each directive is composed of one of the following: one or more white-space wide characters; an ordinary wide character (neither '`%`' nor a white-space wide character); or a conversion specification. It is unspecified whether an encoding error occurs if the format string contains **wchar_t** values that do not correspond to members of the character set of the current locale and the specified semantics do not require that value to be processed by *wcrtomb*( ).

CX Each conversion specification is introduced by the '`%`' or by the character sequence "`%`*n*`$`", after which the following appear in sequence:

- An optional assignment-suppressing character `'*'`.
- An optional non-zero decimal integer that specifies the maximum field width.
- CX An optional assignment-allocation character `'m'`.
- An optional length modifier that specifies the size of the receiving object.
- A conversion specifier wide character that specifies the type of conversion to be applied. The valid conversion specifiers are described below.

The *fwscanf*( ) functions shall execute each directive of the format in turn. When all directives have been executed, or if a directive fails (as detailed below), the function shall return. Failures are described as input failures (due to the unavailability of input bytes) or matching failures (due to inappropriate input).

A directive composed of one or more white-space wide characters shall be executed by reading input up to the first wide character that is not a white-space wide character, which shall remain unread, or until no more wide characters can be read. The directive shall never fail.

A directive that is an ordinary wide character shall be executed as follows. The next wide character is read from the input and compared with the wide character that comprises the directive; if the comparison shows that they are not equivalent, the directive shall fail, and the differing and subsequent wide characters remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a wide character from being read, the directive shall fail.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each conversion wide character. A conversion specification is executed in the following steps.

Input white-space wide characters shall be skipped, unless the conversion specification includes a `[`, `c`, or `n` conversion specifier.

An item shall be read from the input, unless the conversion specification includes an `n` conversion specifier wide character. An input item is defined as the longest sequence of input wide characters, not exceeding any specified field width, which is an initial subsequence of a matching sequence. The first wide character, if any, after the input item shall remain unread. If the length of the input item is zero, the execution of the conversion specification shall fail; this condition is a matching failure, unless end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is an input failure.

Except in the case of a `%` conversion specifier, the input item (or, in the case of a `%n` conversion specification, the count of input wide characters) shall be converted to a type appropriate to the conversion wide character. If the input item is not a matching sequence, the execution of the conversion specification shall fail; this condition is a matching failure. Unless assignment suppression was indicated by a `'*'`, the result of the conversion shall be placed in the object pointed to by the first argument following the *format* argument that has not already received a conversion result if the conversion specification is introduced by `%`, CX or in the *n*th argument if introduced by the wide-character sequence `"%`*n*`$"`. If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.

CX The `c`, `s`, and `[` conversion specifiers shall accept an optional assignment-allocation character `'m'`, which shall cause a memory buffer to be allocated to hold the conversion results. If the conversion specifier is `s` or `[`, the allocated buffer shall include space for a terminating null character (or wide character). In such a case, the argument corresponding to the conversion specifier should be a reference to a pointer value that will receive a pointer to the allocated buffer. The system shall allocate a buffer as if *malloc*( ) had been called. The application shall be

responsible for freeing the memory after usage. If there is insufficient memory to allocate a buffer, the function shall set *errno* to [ENOMEM] and a conversion error shall result. If the function returns EOF, any memory successfully allocated for parameters using assignment-allocation character `'m'` by this call shall be freed before the function returns.

The length modifiers and their meanings are:

`hh` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **signed char** or **unsigned char**.

`h` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **short** or **unsigned short**.

`l` (ell) Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **long** or **unsigned long**; that a following `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier applies to an argument with type pointer to **double**; or that a following `c`, `s`, or `[` conversion specifier applies to an argument with type pointer to **wchar_t**. If the `'m'` assignment-allocation character is specified, the conversion applies to an argument with the type pointer to a pointer to **wchar_t**.

CX

`ll` (ell-ell) Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **long long** or **unsigned long long**.

`j` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **intmax_t** or **uintmax_t**.

`z` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **size_t** or the corresponding signed integer type.

`t` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **ptrdiff_t** or the corresponding **unsigned** type.

`L` Specifies that a following `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier applies to an argument with type pointer to **long double**.

If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.

The following conversion specifier wide characters are valid:

`d` Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of *wcstol*() with the value 10 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **int**.

`i` Matches an optionally signed integer, whose format is the same as expected for the subject sequence of *wcstol*() with 0 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **int**.

`o` Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of *wcstoul*() with the value 8 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **unsigned**.

`u` Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of *wcstoul*() with the value 10 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a

pointer to **unsigned**.

`x` Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of *wcstoul*( ) with the value 16 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **unsigned**.

`a,e,f,g`
Matches an optionally signed floating-point number, infinity, or NaN whose format is the same as expected for the subject sequence of *wcstod*( ). In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **float**.

If the *fwprintf*( ) family of functions generates character string representations for infinity and NaN (a symbolic entity encoded in floating-point format) to support IEEE Std 754-1985, the *fwscanf*( ) family of functions shall recognize them as input.

`s` Matches a sequence of non-white-space wide characters. If no `l` (ell) qualifier is present, characters from the input field shall be converted as if by repeated calls to the *wcrtomb*( ) function, with the conversion state described by an **mbstate_t** object initialized to zero before the first wide character is converted. If the `'m'` assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to a character array large enough to accept the sequence and the
CX terminating null character, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a **char**.

If the `l` (ell) qualifier is present and the `'m'` assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of **wchar_t** large enough to accept the sequence and the terminating null wide
CX character, which shall be added automatically. If the `l` (ell) qualifier is present and the `'m'` assignment-allocation character is present, the application shall ensure that the corresponding argument is a pointer to a pointer to a **wchar_t**.

`[` Matches a non-empty sequence of wide characters from a set of expected wide characters (the *scanset*). If no `l` (ell) qualifier is present, wide characters from the input field shall be converted as if by repeated calls to the *wcrtomb*( ) function, with the conversion state described by an **mbstate_t** object initialized to zero before the first wide character is converted. If the `'m'` assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to a character array large enough to accept the sequence and the terminating null character, which
CX shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a **char**.

If an `l` (ell) qualifier is present and the `'m'` assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of **wchar_t** large enough to accept the sequence and the terminating null wide
CX character. If an `l` (ell) qualifier is present and the `'m'` assignment-allocation character is specified, the application shall ensure that the corresponding argument is a pointer to a pointer to a **wchar_t**.

The conversion specification includes all subsequent wide characters in the *format* string up to and including the matching <right-square-bracket> (`']'`). The wide characters between the square brackets (the *scanlist*) comprise the scanset, unless the wide character after the <left-square-bracket> is a <circumflex> (`'^'`), in which case the scanset contains all wide characters that do not appear in the scanlist between the

<circumflex> and the <right-square-bracket>. If the conversion specification begins with `"[]"` or `"[^]"`, the <right-square-bracket> is included in the scanlist and the next <right-square-bracket> is the matching <right-square-bracket> that ends the conversion specification; otherwise, the first <right-square-bracket> is the one that ends the conversion specification. If a `'-'` is in the scanlist and is not the first wide character, nor the second where the first wide character is a `'^'`, nor the last wide character, the behavior is implementation-defined.

`c` Matches a sequence of wide characters of exactly the number specified by the field width (1 if no field width is present in the conversion specification).

If no `l` (ell) length modifier is present, characters from the input field shall be converted as if by repeated calls to the *wcrtomb*( ) function, with the conversion state described by an **mbstate_t** object initialized to zero before the first wide character is converted. No null character is added. If the `'m'` assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to the initial element of a character array large enough to accept the sequence. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a **char**.

No null wide character is added. If an `l` (ell) length modifier is present and the `'m'` assignment-allocation character is not specified, the application shall ensure that the corresponding argument shall be a pointer to the initial element of an array of **wchar_t** large enough to accept the sequence. If an `l` (ell) qualifier is present and the `'m'` assignment-allocation character is specified, the application shall ensure that the corresponding argument is a pointer to a pointer to a **wchar_t**.

`p` Matches an implementation-defined set of sequences, which shall be the same as the set of sequences that is produced by the `%p` conversion specification of the corresponding *fwprintf*( ) functions. The application shall ensure that the corresponding argument is a pointer to a pointer to **void**. The interpretation of the input item is implementation-defined. If the input item is a value converted earlier during the same program execution, the pointer that results shall compare equal to that value; otherwise, the behavior of the `%p` conversion is undefined.

`n` No input is consumed. The application shall ensure that the corresponding argument is a pointer to the integer into which is to be written the number of wide characters read from the input so far by this call to the *fwscanf*( ) functions. Execution of a `%n` conversion specification shall not increment the assignment count returned at the completion of execution of the function. No argument shall be converted, but one shall be consumed. If the conversion specification includes an assignment-suppressing wide character or a field width, the behavior is undefined.

XSI `C` Equivalent to `lc`.

XSI `S` Equivalent to `ls`.

`%` Matches a single `'%'` wide character; no conversion or assignment shall occur. The complete conversion specification shall be `%%`.

If a conversion specification is invalid, the behavior is undefined.

The conversion specifiers `A`, `E`, `F`, `G`, and `X` are also valid and shall be equivalent to, respectively, `a`, `e`, `f`, `g`, and `x`.

If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any wide characters matching the current conversion specification (except for `%n`) have been

read (other than leading white-space wide characters, where permitted), execution of the current conversion specification shall terminate with an input failure. Otherwise, unless execution of the current conversion specification is terminated with a matching failure, execution of the following conversion specification (if any) shall be terminated with an input failure.

Reaching the end of the string in *swscanf*( ) shall be equivalent to encountering end-of-file for *fwscanf*( ).

If conversion terminates on a conflicting input, the offending input shall be left unread in the input. Any trailing white-space wide characters (including <newline>) shall be left unread unless matched by a conversion specification. The success of literal matches and suppressed assignments is only directly determinable via the `%n` conversion specification.

CX The *fwscanf*( ) and *wscanf*( ) functions may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetwc*( ), *fgetws*( ), *fwscanf*( ), *getwc*( ), *getwchar*( ), *vfwscanf*( ), *vwscanf*( ), or *wscanf*( ) using *stream* that returns data not supplied by a prior call to *ungetwc*( ).

RETURN VALUE

Upon successful completion, these functions shall return the number of successfully matched and assigned input items; this number can be zero in the event of an early matching failure. If the input ends before the first conversion (if any) has completed, and without a matching failure having occurred, EOF shall be returned. If an error occurs before the first conversion (if any) has
CX completed, and without a matching failure having occurred, EOF shall be returned and *errno* shall be set to indicate the error. If a read error occurs, the error indicator for the stream shall be set.

ERRORS

For the conditions under which the *fwscanf*( ) functions shall fail and may fail, refer to *fgetwc*( ).

In addition, the *fwscanf*( ) function shall fail if:

CX [EILSEQ] Input byte sequence does not form a valid character.

[ENOMEM] Insufficient storage space is available.

In addition, the *fwscanf*( ) function may fail if:

CX [EINVAL] There are insufficient arguments.

EXAMPLES

The call:

```
int i, n; float x; char name[50];
n = wscanf(L"%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 Hamster
```

assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string `"Hamster"`.

The call:

```
int i; float x; char name[50];
(void) wscanf(L"%2d%f%*d %[0123456789]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```


assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string `"56\0"` in *name*. The next call to *getchar*( ) shall return the character `'a'`.

APPLICATION USAGE

In format strings containing the `'%'` form of conversion specifications, each argument in the argument list is used exactly once.

For functions that allocate memory as if by *malloc*( ), the application should release such memory when it is no longer required by a call to *free*( ). For *fwscanf*( ), this is memory allocated via use of the `'m'` assignment-allocation character.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *getwc*( ), *fwprintf*( ), *setlocale*( ), *wcstod*( ), *wcstol*( ), *wcstoul*( ), *wcrtomb*( )

XBD Chapter 7 (on page 127), **<inttypes.h>**, **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899: 1990/Amendment 1: 1995 (E).

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The prototypes for *fwscanf*( ) and *swscanf*( ) are updated.
- The DESCRIPTION is updated.
- The `hh`, `ll`, `j`, `t`, and `z` length modifiers are added.
- The `a`, `A`, and `F` conversion characters are added.

The DESCRIPTION is updated to use the terms ``conversion specifier'' and ``conversion specification'' consistently.

Issue 7

Austin Group Interpretation 1003.1-2001 #170 is applied.

SD5-XSH-ERN-132 is applied, adding the assignment-allocation character `'m'`.

Functionality relating to the `"%n$"` form of conversion specification is moved from the XSI option to the Base.

Changes are made related to support for finegrained timestamps.

The APPLICATION USAGE section is updated to clarify that memory is allocated as if by *malloc*( ).

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0229 [302] and XSH/TC1-2008/0230 [14] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0148 [73], XSH/TC2-2008/0149 [823], and XSH/TC2-2008/0150 [936] are applied.

Issue 8

Austin Group Defect 1163 is applied, clarifying the handling of white space in the format string.

Austin Group Defect 1173 is applied, clarifying the description of the assignment-allocation character `'m'`.

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018 standard.

Austin Group Defect 1375 is applied, changing ``terminating null wide character'' to ``terminating null character (or wide character)'' and changing the first occurrence of **wchar_t** in the descriptions of the `s` and `[` conversion specifiers to **char**.

NAME

gai_strerror — address and name information error description

SYNOPSIS

```
#include <netdb.h>

const char *gai_strerror(int ecode);
```

DESCRIPTION

The *gai_strerror*( ) function shall return a text string describing an error value for the *getaddrinfo*( ) and *getnameinfo*( ) functions listed in the **<netdb.h>** header.

When the *ecode* argument is one of the following values listed in the **<netdb.h>** header:

[EAI_AGAIN]	[EAI_NONAME]
[EAI_BADFLAGS]	[EAI_OVERFLOW]
[EAI_FAIL]	[EAI_SERVICE]
[EAI_FAMILY]	[EAI_SOCKTYPE]
[EAI_MEMORY]	[EAI_SYSTEM]

the function return value shall point to a string describing the error. If the argument is not one of those values, the function shall return a pointer to a string whose contents indicate an unknown error.

RETURN VALUE

Upon successful completion, *gai_strerror*( ) shall return a pointer to an implementation-defined string.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

freeaddrinfo( )

XBD **<netdb.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The Open Group Base Resolution bwg2001-009 is applied, which changes the return type from **char** * to **const char** *. This is for coordination with the IPnG Working Group.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/22 is applied, adding the [EAI_OVERFLOW] error code.

NAME

getaddrinfo — get address information

SYNOPSIS

```
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(const char *restrict nodename,
    const char *restrict servname,
    const struct addrinfo *restrict hints,
    struct addrinfo **restrict res);
```

DESCRIPTION

Refer to *freeaddrinfo( )*.

NAME

getc — get a byte from a stream

SYNOPSIS

```
#include <stdio.h>

int getc(FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *getc*( ) function shall be equivalent to *fgetc*( ), except that if it is implemented as a macro it may evaluate *stream* more than once, so the argument should never be an expression with side-effects.

RETURN VALUE

Refer to *fgetc*( ).

ERRORS

Refer to *fgetc*( ).

EXAMPLES

None.

APPLICATION USAGE

If the integer value returned by *getc*( ) is stored into a variable of type **char** and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a variable of type **char** on widening to integer is implementation-defined.

Since it may be implemented as a macro, *getc*( ) may treat incorrectly a *stream* argument with side-effects. In particular, *getc*(*f*++) does not necessarily work as expected. Therefore, use of this function should be preceded by `"#undef getc"` in such situations; *fgetc*( ) could also be used.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fgetc*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0231 [14] is applied.

NAME

getc_unlocked, getchar_unlocked, putc_unlocked, putchar_unlocked — stdio with explicit client locking

SYNOPSIS

CX
```
#include <stdio.h>

int getc_unlocked(FILE *stream);
int getchar_unlocked(void);
int putc_unlocked(int c, FILE *stream);
int putchar_unlocked(int c);
```

DESCRIPTION

Versions of the functions *getc*(), *getchar*(), *putc*(), and *putchar*() respectively named *getc_unlocked*(), *getchar_unlocked*(), *putc_unlocked*(), and *putchar_unlocked*() shall be provided which are functionally equivalent to the original versions, with the exception that they are not required to be implemented in a fully thread-safe manner. They shall be thread-safe when used within a scope protected by *flockfile*() (or *ftrylockfile*()) and *funlockfile*(). These functions can safely be used in a multi-threaded program if and only if they are called while the invoking thread owns the (**FILE** *) object, as is the case after a successful call to the *flockfile*() or *ftrylockfile*() functions.

If *getc_unlocked*() or *putc_unlocked*() are implemented as macros they may evaluate *stream* more than once, so the *stream* argument should never be an expression with side-effects.

RETURN VALUE

See *getc*(), *getchar*(), *putc*(), and *putchar*().

ERRORS

See *getc*(), *getchar*(), *putc*(), and *putchar*().

EXAMPLES

None.

APPLICATION USAGE

Since they may be implemented as macros, *getc_unlocked*() and *putc_unlocked*() may treat incorrectly a *stream* argument with side-effects. In particular, *getc_unlocked*(*f++) and *putc_unlocked*(c,*f++) do not necessarily work as expected. Therefore, use of these functions in such situations should be preceded by the following statement as appropriate:

```
#undef getc_unlocked
#undef putc_unlocked
```

RATIONALE

Some I/O functions are typically implemented as macros for performance reasons (for example, *putc*() and *getc*()). For safety, they need to be synchronized, but it is often too expensive to synchronize on every character. Nevertheless, it was felt that the safety concerns were more important; consequently, the *getc*(), *getchar*(), *putc*(), and *putchar*() functions are required to be thread-safe. However, unlocked versions are also provided with names that clearly indicate the unsafe nature of their operation but can be used to exploit their higher performance. These unlocked versions can be safely used only within explicitly locked program regions, using exported locking primitives. In particular, a sequence such as:

```
flockfile(fileptr);
putc_unlocked('1', fileptr);
putc_unlocked('\n', fileptr);
fprintf(fileptr, "Line 2\n");
```



```
funlockfile(fileptr);
```

is permissible, and results in the text sequence:

```
1
Line 2
```

being printed without being interspersed with output from other threads.

It would be wrong to have the standard names such as *getc*( ), *putc*( ), and so on, map to the ``faster, but unsafe'' rather than the ``slower, but safe'' versions. In either case, you would still want to inspect all uses of *getc*( ), *putc*( ), and so on, by hand when converting existing code. Choosing the safe bindings as the default, at least, results in correct code and maintains the ``atomicity at the function'' invariant. To do otherwise would introduce gratuitous synchronization errors into converted code. Other routines that modify the *stdio* (**FILE** *) structures or buffers are also safely synchronized.

Note that there is no need for functions of the form *getc_locked*( ), *putc_locked*( ), and so on, since this is the functionality of *getc*( ), *putc*( ), *et al*. It would be inappropriate to use a feature test macro to switch a macro definition of *getc*( ) between *getc_locked*( ) and *getc_unlocked*( ), since the ISO C standard requires an actual function to exist, a function whose behavior could not be changed by the feature test macro. Also, providing both the *xxx_locked*( ) and *xxx_unlocked*( ) forms leads to the confusion of whether the suffix describes the behavior of the function or the circumstances under which it should be used.

Three additional routines, *flockfile*( ), *ftrylockfile*( ), and *funlockfile*( ) (which may be macros), are provided to allow the user to delineate a sequence of I/O statements that are executed synchronously.

The *ungetc*( ) function is infrequently called relative to the other functions/macros so no unlocked variation is needed.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *flockfile*( ), *getc*( ), *getchar*( ), *putc*( ), *putchar*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

These functions are marked as part of the Thread-Safe Functions option.

The Open Group Corrigendum U030/2 is applied, adding APPLICATION USAGE describing how applications should be written to avoid the case when the functions are implemented as macros.

Issue 7

The *getc_unlocked*( ), *getchar_unlocked*( ), *putc_unlocked*( ), and *putchar_unlocked*( ) functions are moved from the Thread-Safe Functions option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0232 [395], XSH/TC1-2008/0233 [395], XSH/TC1-2008/0234 [395], and XSH/TC1-2008/0235 [14] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0151 [826] is applied.

NAME

getchar — get a byte from a *stdin* stream

SYNOPSIS

```
#include <stdio.h>

int getchar(void);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *getchar*( ) function shall be equivalent to *getc*(*stdin*).

RETURN VALUE

Refer to *fgetc*( ).

ERRORS

Refer to *fgetc*( ).

EXAMPLES

None.

APPLICATION USAGE

If the integer value returned by *getchar*( ) is stored into a variable of type **char** and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a variable of type **char** on widening to integer is implementation-defined.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *getc*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0236 [14] is applied.

37926 **NAME**

37927 getchar_unlocked — stdio with explicit client locking

37928 **SYNOPSIS**

37929 CX
```
#include <stdio.h>

int getchar_unlocked(void);
```

37931 **DESCRIPTION**

37932 Refer to *getc_unlocked( )*.

37933 **NAME**

37934 getcwd — get the pathname of the current working directory

37935 **SYNOPSIS**

37936 ```
#include <unistd.h>
```

37937 ```
char *getcwd(char *buf, size_t size);
```

37938 **DESCRIPTION**

37939 The *getcwd*( ) function shall place an absolute pathname of the current working directory in the
37940 array pointed to by *buf*, and return *buf*. The pathname shall contain no components that are dot
37941 or dot-dot, or are symbolic links.

37942 If there are multiple pathnames that *getcwd*( ) could place in the array pointed to by *buf*, one
37943 beginning with a single <slash> character and one or more beginning with two <slash>
37944 characters, then *getcwd*( ) shall place the pathname beginning with a single <slash> character in
37945 the array. The pathname shall not contain any unnecessary <slash> characters after the leading
37946 one or two <slash> characters.

37947 The *size* argument is the size in bytes of the character array pointed to by the *buf* argument. If *buf*
37948 is a null pointer, the behavior of *getcwd*( ) is unspecified.

37949 **RETURN VALUE**

37950 Upon successful completion, *getcwd*( ) shall return the *buf* argument. Otherwise, *getcwd*( ) shall
37951 return a null pointer and set *errno* to indicate the error. The contents of the array pointed to by
37952 *buf* are then undefined.

37953 **ERRORS**

37954 The *getcwd*( ) function shall fail if:

37955 [EINVAL] The *size* argument is 0.

37956 [ERANGE] The *size* argument is greater than 0, but is smaller than the length of the string
37957 +1.

37958 The *getcwd*( ) function may fail if:

37959 [EACCES] Search permission was denied for the current directory, or read or search
37960 permission was denied for a directory above the current directory in the file
37961 hierarchy.

37962 [ENOMEM] Insufficient storage space is available.

37963 **EXAMPLES**

37964 The following example uses {PATH_MAX} as the initial buffer size (unless it is indeterminate or
37965 very large), and calls *getcwd*( ) with progressively larger buffers until it does not give an
37966 [ERANGE] error.

37967 ```
#include <stdlib.h>
37968 #include <errno.h>
37969 #include <unistd.h>

37970 ...

37971 long path_max;
37972 size_t size;
37973 char *buf;
37974 char *ptr;

37975 path_max = pathconf(".", _PC_PATH_MAX);
37976 if (path_max == -1)
```

  
```

```
            size = 1024;
        else if (path_max > 10240)
            size = 10240;
        else
            size = path_max;

        for (buf = ptr = NULL; ptr == NULL; size *= 2)
        {
            if ((buf = realloc(buf, size)) == NULL)
            {
                ... handle error ...
            }

            ptr = getcwd(buf, size);
            if (ptr == NULL && errno != ERANGE)
            {
                ... handle error ...
            }
        }
        ...
        free (buf);
```

APPLICATION USAGE

If the pathname obtained from *getcwd*( ) is longer than {PATH_MAX} bytes, it could produce an [ENAMETOOLONG] error if passed to *chdir*( ). Therefore, in order to return to that directory it may be necessary to break the pathname into sections shorter than {PATH_MAX} bytes and call *chdir*( ) on each section in turn (the first section being an absolute pathname and subsequent sections being relative pathnames). A simpler way to handle saving and restoring the working directory when it may be deeper than {PATH_MAX} bytes in the file hierarchy is to use a file descriptor and *fchdir*( ), rather than *getcwd*( ) and *chdir*( ). However, the two methods do have some differences. The *fchdir*( ) approach causes the program to restore a working directory even if it has been renamed in the meantime, whereas the *chdir*( ) approach restores to a directory with the same name as the original, even if the directories were renamed in the meantime. Since the *fchdir*( ) approach does not access parent directories, it can succeed when *getcwd*( ) would fail due to permissions problems. In applications conforming to earlier versions of this standard, it was not possible to use the *fchdir*( ) approach when the working directory is searchable but not readable, as the only way to open a directory was with O_RDONLY, whereas the *getcwd*( ) approach can succeed in this case.

RATIONALE

Having *getcwd*( ) take no arguments and instead use the *malloc*( ) function to produce space for the returned argument was considered. The advantage is that *getcwd*( ) knows how big the working directory pathname is and can allocate an appropriate amount of space. But the programmer would have to use the *free*( ) function to free the resulting object, or each use of *getcwd*( ) would further reduce the available memory. Finally, *getcwd*( ) is taken from the SVID where it has the two arguments used in this volume of POSIX.1-2024.

The older function *getwd*( ) was rejected for use in this context because it had only a buffer argument and no *size* argument, and thus had no way to prevent overwriting the buffer, except to depend on the programmer to provide a large enough buffer.

On some implementations, if *buf* is a null pointer, *getcwd*( ) may obtain *size* bytes of memory using *malloc*( ). In this case, the pointer returned by *getcwd*( ) may be used as the argument in a subsequent call to *free*( ). Invoking *getcwd*( ) with *buf* as a null pointer is not recommended in conforming applications.

Earlier implementations of *getcwd*() sometimes generated pathnames like `"../../../subdirname"` internally, using them to explore the path of ancestor directories back to the root. If one of these internal pathnames exceeded {PATH_MAX} in length, the implementation could fail with *errno* set to [ENAMETOOLONG]. This is no longer allowed.

If a program is operating in a directory where some (grand)parent directory does not permit reading, *getcwd*() may fail, as in most implementations it must read the directory to determine the name of the file. This can occur if search, but not read, permission is granted in an intermediate directory, or if the program is placed in that directory by some more privileged process (for example, login). Including the [EACCES] error condition makes the reporting of the error consistent and warns the application developer that *getcwd*() can fail for reasons beyond the control of the application developer or user. Some implementations can avoid this occurrence (for example, by implementing *getcwd*() using *pwd*, where *pwd* is a set-user-root process), thus the error was made optional. Since this volume of POSIX.1-2024 permits the addition of other errors, this would be a common addition and yet one that applications could not be expected to deal with without this addition.

FUTURE DIRECTIONS

None.

SEE ALSO

malloc( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [ENOMEM] optional error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #140 is applied, changing the text for consistency with the *pwd* utility, adding text to address the case where the current directory is deeper in the file hierarchy than {PATH_MAX} bytes, and adding the requirements relating to pathnames beginning with two <slash> characters.

NAME

getdate — convert user format date and time

SYNOPSIS

XSI

```
#include <time.h>

struct tm *getdate(const char *string);
```

DESCRIPTION

The *getdate*( ) function shall convert a string representation of a date or time into a broken-down time.

The external variable or macro *getdate_err*, which has type **int**, is used by *getdate*( ) to return error values. It is unspecified whether *getdate_err* is a macro or an identifier declared with external linkage, and whether or not it is a modifiable lvalue. If a macro definition is suppressed in order to access an actual object, or a program defines an identifier with the name *getdate_err*, the behavior is undefined.

Templates are used to parse and interpret the input string. The templates are contained in a text file identified by the environment variable *DATEMSK*. The *DATEMSK* variable should be set to indicate the full pathname of the file that contains the templates. The first line in the template that matches the input specification is used for interpretation and conversion into the internal time format.

The following conversion specifications shall be supported:

`%%` Equivalent to `%`.

`%a` Abbreviated weekday name.

`%A` Full weekday name.

`%b` Abbreviated month name.

`%B` Full month name.

`%c` Locale's appropriate date and time representation.

`%C` Century number [00,99]; leading zeros are permitted but not required.

`%d` Day of month [01,31]; the leading 0 is optional.

`%D` Date as `%m/%d/%y`.

`%e` Equivalent to `%d`.

`%h` Abbreviated month name.

`%H` Hour [00,23].

`%I` Hour [01,12].

`%m` Month number [01,12].

`%M` Minute [00,59].

`%n` Equivalent to <newline>.

`%p` Locale's equivalent of either AM or PM.

`%r` The locale's appropriate representation of time in 12-hour clock notation, if the 12-hour format is supported in the locale (see XBD Section 7.3.5, on page 152). In the POSIX locale, this shall be equivalent to `%I:%M:%S %p`.

38096 `%R` Time as `%H:%M`.

38097 `%S` Seconds [00,60]. The range goes to 60 (rather than stopping at 59) to allow positive leap
38098 seconds to be expressed. Since leap seconds cannot be predicted by any algorithm, leap
38099 second data has to come from some external source.

38100 `%t` Equivalent to <tab>.

38101 `%T` Time as `%H:%M:%S`.

38102 `%w` Weekday number (Sunday = [0,6]).

38103 `%x` Locale's appropriate date representation.

38104 `%X` Locale's appropriate time representation.

38105 `%y` Year within century. When a century is not otherwise specified, values in the range
38106 [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall
38107 refer to years 2000 to 2068 inclusive.

38108 **Note:** It is expected that in a future version of this standard the default century inferred
38109 from a 2-digit year will change. (This would apply to all commands accepting a
38110 2-digit year as input.)

38111 `%Y` Year as `"ccyy"` (for example, 2001).

38112 `%Z` Timezone name or no characters if no timezone exists. If the timezone supplied by `%Z` is
38113 not the timezone that *getdate*( ) expects, an invalid input specification error shall result.
38114 The *getdate*( ) function calculates an expected timezone based on information supplied
38115 to the function (such as the hour, day, and month).

38116 The match between the template and input specification performed by *getdate*( ) shall be case-
38117 insensitive.

38118 The month and weekday names can consist of any combination of upper and lowercase letters.
38119 The process can request that the input date or time specification be in a specific language by
38120 setting the *LC_TIME* category (see *setlocale*( )).

38121 Leading zeros are not necessary for the descriptors that allow leading zeros. However, at most
38122 two digits are allowed for those descriptors, including leading zeros. Extra white space in either
38123 the template file or in *string* shall be ignored.

38124 The results are undefined if the conversion specifications `%c`, `%x`, and `%X` include unsupported
38125 conversion specifications.

38126 The following rules apply for converting the input specification into the internal format:

38127 • If `%Z` is being scanned, then *getdate*( ) shall initialize the broken-down time to be the current
38128 time in the scanned timezone. Otherwise, it shall initialize the broken-down time based on
38129 the current local time as if *localtime*( ) had been called.

38130 • If only the weekday is given, the day chosen shall be the day, starting with today and
38131 moving into the future, which first matches the named day.

38132 • If only the month (and no year) is given, the month chosen shall be the month, starting
38133 with the current month and moving into the future, which first matches the named month.
38134 The first day of the month shall be assumed if no day is given.

38135 • If no hour, minute, and second are given, the current hour, minute, and second shall be
38136 assumed.

38137 • If no date is given, the hour chosen shall be the hour, starting with the current hour and
38138 moving into the future, which first matches the named hour.

38139 If a conversion specification in the DATEMSK file does not correspond to one of the conversion
38140 specifications above, the behavior is unspecified.

38141 The *getdate*( ) function need not be thread-safe.

38142 **RETURN VALUE**

38143 Upon successful completion, *getdate*( ) shall return a pointer to a **struct tm**. Otherwise, it shall
38144 return a null pointer and set *getdate_err* to indicate the error.

38145 **ERRORS**

38146 The *getdate*( ) function shall fail in the following cases, setting *getdate_err* to the value shown in
38147 the list below. Any changes to *errno* are unspecified.

38148 1. The *DATEMSK* environment variable is null or undefined.

38149 2. The template file cannot be opened for reading.

38150 3. Failed to get file status information.

38151 4. The template file is not a regular file.

38152 5. An I/O error is encountered while reading the template file.

38153 6. Memory allocation failed (not enough memory available).

38154 7. There is no line in the template that matches the input.

38155 8. Invalid input specification. For example, February 31; or a time is specified that cannot be
38156 represented in a **time_t** (representing the time in seconds since the Epoch).

38157 **EXAMPLES**

38158 1. The following example shows the possible contents of a template:

```
38159 %m
38160 %A %B %d, %Y, %H:%M:%S
38161 %A
38162 %B
38163 %m/%d/%y %I %p
38164 %d,%m,%Y %H:%M
38165 at %A the %dst of %B in %Y
38166 run job at %I %p,%B %dnd
38167 %A den %d. %B %Y %H.%M Uhr
```

38168 2. The following are examples of valid input specifications for the template in Example 1:

```
38169 getdate("10/1/87 4 PM");
38170 getdate("Friday");
38171 getdate("Friday September 18, 1987, 10:30:30");
38172 getdate("24,9,1986 10:30");
38173 getdate("at monday the 1st of december in 1986");
38174 getdate("run job at 3 PM, december 2nd");
```

38175 If the *LC_TIME* category is set to a German locale that includes *freitag* as a weekday name
38176 and *oktober* as a month name, the following would be valid:

```
38177 getdate("freitag den 10. oktober 1986 10.30 Uhr");
```


3. The following example shows how local date and time specification can be defined in the template:

Invocation	Line in Template
`getdate("11/27/86")`	`%m/%d/%y`
`getdate("27.11.86")`	`%d.%m.%y`
`getdate("86-11-27")`	`%y-%m-%d`
`getdate("Friday 12:00:00")`	`%A %H:%M:%S`

4. The following examples help to illustrate the above rules assuming that the current date is Mon Sep 22 12:19:47 EDT 1986 and the *LC_TIME* category is set to the default C or POSIX locale:

Input	Line in Template	Date
`Mon`	`%a`	Mon Sep 22 12:19:47 EDT 1986
`Sun`	`%a`	Sun Sep 28 12:19:47 EDT 1986
`Fri`	`%a`	Fri Sep 26 12:19:47 EDT 1986
`September`	`%B`	Mon Sep 1 12:19:47 EDT 1986
`January`	`%B`	Thu Jan 1 12:19:47 EST 1987
`December`	`%B`	Mon Dec 1 12:19:47 EST 1986
`Sep Mon`	`%b %a`	Mon Sep 1 12:19:47 EDT 1986
`Jan Fri`	`%b %a`	Fri Jan 2 12:19:47 EST 1987
`Dec Mon`	`%b %a`	Mon Dec 1 12:19:47 EST 1986
`Jan Wed 1989`	`%b %a %Y`	Wed Jan 4 12:19:47 EST 1989
`Fri 9`	`%a %H`	Fri Sep 26 09:00:00 EDT 1986
`Feb 10:30`	`%b %H:%S`	Sun Feb 1 10:00:30 EST 1987
`10:30`	`%H:%M`	Tue Sep 23 10:30:00 EDT 1986
`13:30`	`%H:%M`	Mon Sep 22 13:30:00 EDT 1986

APPLICATION USAGE

Although historical versions of *getdate*( ) did not require that **<time.h>** declare the external variable *getdate_err*, this volume of POSIX.1-2024 does require it. The standard developers encourage applications to remove declarations of *getdate_err* and instead incorporate the declaration by including **<time.h>**.

Applications should use `%Y` (4-digit years) in preference to `%y` (2-digit years).

RATIONALE

In standard locales, the conversion specifications `%c`, `%x`, and `%X` do not include unsupported conversion specifiers and so the text regarding results being undefined is not a problem in that case.

FUTURE DIRECTIONS

None.

SEE ALSO

ctime( ), *localtime*( ), *setlocale*( ), *strftime*( ), *times*( )

XBD **<time.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The last paragraph of the DESCRIPTION is added.

The `%C` conversion specification is added, and the exact meaning of the `%y` conversion specification is clarified in the DESCRIPTION.

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

The `%R` conversion specification is changed to follow historical practice.

Issue 6

The DESCRIPTION is updated to refer to ``seconds since the Epoch'' rather than ``seconds since 00:00:00 UTC (Coordinated Universal Time), January 1 1970'' for consistency with other *time* functions.

The description of `%S` is updated so that the valid range is [00,60] rather than [00,61].

The DESCRIPTION is updated to refer to conversion specifications instead of field descriptors for consistency with other functions.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

The description of the *getdate_err* value is expanded.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0152 [796] is applied.

Issue 8

Austin Group Defect 1307 is applied, changing the `%r` conversion in relation to locales that do not support the 12-hour clock format.

NAME

getdelim, getline — read a delimited record from *stream*

SYNOPSIS

CX
```
#include <stdio.h>

ssize_t getdelim(char **restrict lineptr, size_t *restrict n,
    int delimiter, FILE *restrict stream);
ssize_t getline(char **restrict lineptr, size_t *restrict n,
    FILE *restrict stream);
```

DESCRIPTION

The *getdelim*( ) function shall read from *stream* until it encounters a character matching the *delimiter* character. The *delimiter* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** of equal value that terminates the read process. If the *delimiter* argument has any other value, the behavior is undefined.

The application shall ensure that **lineptr* is a valid argument that could be passed to the *free*( ) function. If **n* is non-zero, the application shall ensure that **lineptr* either points to an object of size at least **n* bytes, or is a null pointer.

If **lineptr* is a null pointer or if the object pointed to by **lineptr* is of insufficient size, an object shall be allocated as if by *malloc*( ) or the object shall be reallocated as if by *realloc*( ), respectively, such that the object is large enough to hold the characters to be written to it, including the terminating NUL, and **n* shall be set to the new size. If the object was allocated, or if the reallocation operation moved the object, **lineptr* shall be updated to point to the new object or new location. The characters read, including any delimiter, shall be stored in the object, and a terminating NUL added when the delimiter or end-of-file is encountered.

The *getline*( ) function shall be equivalent to the *getdelim*( ) function with the *delimiter* character equal to the <newline> character.

The *getdelim*( ) and *getline*( ) functions may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetc*( ), *fgets*( ), *fread*( ), *fscanf*( ), *getc*( ), *getchar*( ), *getdelim*( ), *getline*( ), or *scanf*( ) using *stream* that returns data not supplied by a prior call to *ungetc*( ).

RETURN VALUE

Upon successful completion, the *getline*( ) and *getdelim*( ) functions shall return the number of bytes written into the buffer, including the delimiter character if one was encountered before EOF, but excluding the terminating NUL character. If the end-of-file indicator for the stream is set, or if no characters were read and the stream is at end-of-file, the end-of-file indicator for the stream shall be set and the function shall return –1. If an error occurs, the error indicator for the stream shall be set, and the function shall return –1 and set *errno* to indicate the error.

ERRORS

For the conditions under which the *getdelim*( ) and *getline*( ) functions shall fail and may fail, refer to *fgetc*( ).

In addition, these functions shall fail if:

[EINVAL] *lineptr* or *n* is a null pointer.

[ENOMEM] Insufficient memory is available.

These functions may fail if:

[EOVERFLOW] The number of bytes to be written into the buffer, including the delimiter character (if encountered), would exceed {SSIZE_MAX}.

EXAMPLES

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *fp;
    char *line = NULL;
    size_t len = 0;
    ssize_t read;
    fp = fopen("/etc/motd", "r");
    if (fp == NULL)
        exit(1);
    while ((read = getline(&line, &len, fp)) != -1) {
        printf("Retrieved line of length %zu :\n", read);
        printf("%s", line);
    }
    if (ferror(fp)) {
        /* handle error */
    }
    free(line);
    fclose(fp);
    return 0;
}
```

APPLICATION USAGE

Setting **lineptr* to a null pointer and **n* to zero are allowed and a recommended way to start parsing a file.

The *ferror*( ) or *feof*( ) functions should be used to distinguish between an error condition and an end-of-file condition.

Although a null terminator is always supplied after the line, note that *strlen*(**lineptr*) will be smaller than the return value if the line contains embedded NUL characters.

RATIONALE

These functions are widely used to solve the problem that the *fgets*( ) function has with long lines. The functions automatically enlarge the target buffers if needed. These are especially useful since they reduce code needed for applications.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fgetc*( ), *fgets*( ), *free*( ), *malloc*( ), *realloc*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 7.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0237 [14] is applied.

38328 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0153 [569], XSH/TC2-2008/0154 [571],
38329 and XSH/TC2-2008/0155 [570] are applied.

38330 **Issue 8**

38331 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

38332 Austin Group Defect 1621 is applied, changing ``NUL terminator'' to ``null terminator''.

NAME

getegid — get the effective group ID

SYNOPSIS

```
#include <unistd.h>

gid_t getegid(void);
```

DESCRIPTION

The *getegid*( ) function shall return the effective group ID of the calling process. The *getegid*( ) function shall not modify *errno*.

RETURN VALUE

The *getegid*( ) function shall always be successful and no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

In a conforming environment, *getegid*( ) will always succeed. It is possible for implementations to provide an extension where a process in a non-conforming environment will not be associated with a user or group ID. It is recommended that such implementations return (**gid_t**)–1 and set *errno* to indicate such an environment; doing so does not violate this standard, since such an environment is already an extension.

FUTURE DIRECTIONS

None.

SEE ALSO

geteuid( ), *getgid*( ), *getresgid*( ), *getresuid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setresgid*( ), *setresuid*( ), *setreuid*( ), *setuid*( )

XBD **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0156 [511] is applied.

Issue 8

Austin Group Defect 1344 is applied, adding *getresgid*( ), *getresuid*( ), *setresgid*( ), and *setresuid*( ) to SEE ALSO.

NAME

getentropy — fill a buffer with random bytes

SYNOPSIS

```
#include <unistd.h>

int getentropy(void *buffer, size_t length);
```

DESCRIPTION

The *getentropy*( ) function shall write *length* bytes of data starting at the location pointed to by *buffer*. The output shall be unpredictable high quality random data, generated by a cryptographically secure pseudo-random number generator. The maximum permitted value for the *length* argument is given by the {GETENTROPY_MAX} symbolic constant defined in **<limits.h>**.

A successful call to *getentropy*( ) shall always provide the requested number of bytes of entropy.

RETURN VALUE

Upon successful completion, *getentropy*( ) shall return 0; otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *getentropy*( ) function shall fail if:

[EINVAL] The value of *length* is greater than {GETENTROPY_MAX}.

The *getentropy*( ) function may fail if:

[ENOSYS] The system does not provide the necessary source of entropy.

EXAMPLES

None.

APPLICATION USAGE

The intended use of this function is to create a seed for other pseudo-random number generators.

RATIONALE

The *getentropy*( ) function is not a cancellation point. (See Section 2.9.5.2 (on page 543).)

FUTURE DIRECTIONS

None.

SEE ALSO

drand48( ), *initstate*( ), *rand*( )

XBD **<limits.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 8.

NAME

getenv, secure_getenv — get value of an environment variable

SYNOPSIS

```
#include <stdlib.h>

char *getenv(const char *name);
CX char *secure_getenv(const char *name);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *getenv*( ) function shall search the environment of the calling process (see XBD Chapter 8, on page 167) for the environment variable *name* if it exists and return a pointer to the value of the environment variable. If the specified environment variable cannot be found, a null pointer shall be returned. The application shall ensure that it does not modify the string pointed to by the CX *getenv*( ) function, unless it is part of a modifiable object previously placed in the environment by assigning a new value to *environ* XSI or by using *putenv*( ).

CX The pointer returned by *getenv*( ) shall point to a string within the environment data pointed to by *environ*.

Note: This requirement is an extension to the ISO C standard, which allows *getenv*( ) to copy the data to an internal buffer.

The *secure_getenv*( ) function shall be equivalent to *getenv*( ), except that it shall return a null pointer if the calling process does not meet all of the following security criteria:

1. The effective user ID and real user ID of the calling process were equal during program startup.
2. The effective group ID and real group ID of the calling process were equal during program startup.
3. Additional implementation-defined security criteria.

RETURN VALUE

Upon successful completion, *getenv*( ) shall return a pointer to a string containing the value for the specified *name*. If the specified *name* cannot be found in the environment of the calling process, a null pointer shall be returned.

CX Upon successful completion, *secure_getenv*( ) shall return a pointer to a string containing the value for the specified *name*. If the specified *name* cannot be found in the environment of the calling process, or the calling process does not meet the security criteria listed in DESCRIPTION, a null pointer shall be returned.

ERRORS

No errors are defined.

EXAMPLES

Getting the Value of an Environment Variable

The following example gets the value of the *HOME* environment variable.

```
#include <stdlib.h>
...
const char *name = "HOME";
char *value;

value = getenv(name);
```

APPLICATION USAGE

None.

RATIONALE

The *clearenv*( ) function was considered but rejected. The *putenv*( ) function has now been included for alignment with the Single UNIX Specification.

Some earlier versions of this standard did not require *getenv*( ) to be thread-safe because it was allowed to return a value pointing to an internal buffer. However, this behavior allowed by the ISO C standard is no longer allowed by POSIX.1. POSIX.1 requires the environment data to be available through *environ*[], so there is no reason why *getenv*( ) can't return a pointer to the actual data instead of a copy. Therefore *getenv*( ) is now required to be thread-safe (except when another thread modifies the environment).

Conforming applications are required not to directly modify the pointers to which *environ* points, but to use only the *setenv*( ), *unsetenv*( ), and *putenv*( ) functions, or assignment to *environ* itself, to manipulate the process environment. This constraint allows the implementation to properly manage the memory it allocates. This enables the implementation to free any space it has allocated to strings (and perhaps the pointers to them) stored in *environ* when *unsetenv*( ) is called. A C runtime start-up procedure (that which invokes *main*( ) and perhaps initializes *environ*) can also initialize a flag indicating that none of the environment has yet been copied to allocated storage, or that the separate table has not yet been initialized. If the application switches to a complete new environment by assigning a new value to *environ*, this can be detected by *getenv*( ), *setenv*( ), *unsetenv*( ), or *putenv*( ) and the implementation can at that point reinitialize based on the new environment. (This may include copying the environment strings into a new array and assigning *environ* to point to it.)

In fact, for higher performance of *getenv*( ), implementations that do not provide *putenv*( ) could also maintain a separate copy of the environment in a data structure that could be searched much more quickly (such as an indexed hash table, or a binary tree), and update both it and the linear list at *environ* when *setenv*( ) or *unsetenv*( ) is invoked. On implementations that do provide *putenv*( ), such a copy might still be worthwhile but would need to allow for the fact that applications can directly modify the content of environment strings added with *putenv*( ). For example, if an environment string found by searching the copy is one that was added using *putenv*( ), the implementation would need to check that the string in *environ* still has the same name (and value, if the copy includes values), and whenever searching the copy produces no match the implementation would then need to search each environment string in *environ* that was added using *putenv*( ) in case any of them have changed their names and now match. Thus, each use of *putenv*( ) to add to the environment would reduce the speed advantage of having the copy.

Performance of *getenv*( ) can be important for applications which have large numbers of environment variables. Typically, applications like this use the environment as a resource

database of user-configurable parameters. The fact that these variables are in the user's shell environment usually means that any other program that uses environment variables (such as *ls*, which attempts to use *COLUMNS*), or really almost any utility (*LANG*, *LC_ALL*, and so on) is similarly slowed down by the linear search through the variables.

An implementation that maintains separate data structures, or even one that manages the memory it consumes, is not currently required as it was thought it would reduce consensus among implementors who do not want to change their historical implementations.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *putenv*( ), *setenv*( ), *unsetenv*( )

XBD Chapter 8 (on page 167), **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

Issue 6

The following changes were made to align with the IEEE P1003.1a draft standard:

- References added to the new *setenv*( ) and *unsetenv*( ) functions.

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #062 is applied, clarifying that a call to *putenv*( ) may also cause the string to be overwritten.

Austin Group Interpretation 1003.1-2001 #148 is applied, adding the FUTURE DIRECTIONS.

Austin Group Interpretation 1003.1-2001 #156 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0238 [75,428], XSH/TC1-2008/0239 [167], and XSH/TC1-2008/0240 [167] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0157 [656] is applied.

Issue 8

Austin Group Defects 188 and 1394 are applied, changing *getenv*( ) to be thread-safe.

Austin Group Defect 922 is applied, adding the *secure_getenv*( ) function.

NAME

geteuid — get the effective user ID

SYNOPSIS

```
#include <unistd.h>

uid_t geteuid(void);
```

DESCRIPTION

The *geteuid*( ) function shall return the effective user ID of the calling process. The *geteuid*( ) function shall not modify *errno*.

RETURN VALUE

The *geteuid*( ) function shall always be successful and no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

In a conforming environment, *geteuid*( ) will always succeed. It is possible for implementations to provide an extension where a process in a non-conforming environment will not be associated with a user or group ID. It is recommended that such implementations return (**uid_t**)–1 and set *errno* to indicate such an environment; doing so does not violate this standard, since such an environment is already an extension.

FUTURE DIRECTIONS

None.

SEE ALSO

getegid( ), *getgid*( ), *getresgid*( ), *getresuid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setresgid*( ), *setresuid*( ), *setreuid*( ), *setuid*( )

XBD **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0158 [511] is applied.

38567 **Issue 8**

38568 Austin Group Defect 1344 is applied, adding *getresgid*( ), *getresuid*( ), *setresgid*( ), and *setresuid*( ) to
38569 SEE ALSO.

NAME

getgid — get the real group ID

SYNOPSIS

```
#include <unistd.h>

gid_t getgid(void);
```

DESCRIPTION

The *getgid*( ) function shall return the real group ID of the calling process. The *getgid*( ) function shall not modify *errno*.

RETURN VALUE

The *getgid*( ) function shall always be successful and no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

In a conforming environment, *getgid*( ) will always succeed. It is possible for implementations to provide an extension where a process in a non-conforming environment will not be associated with a user or group ID. It is recommended that such implementations return (**gid_t**)–1 and set *errno* to indicate such an environment; doing so does not violate this standard, since such an environment is already an extension.

FUTURE DIRECTIONS

None.

SEE ALSO

getegid( ), *geteuid*( ), *getresgid*( ), *getresuid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setresgid*( ), *setresuid*( ), *setreuid*( ), *setuid*( )

XBD **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0159 [511] is applied.

38610 **Issue 8**

38611 Austin Group Defect 1344 is applied, adding *getresgid*( ), *getresuid*( ), *setresgid*( ), and *setresuid*( ) to
38612 SEE ALSO.

NAME

getgrent — get the group database entry

SYNOPSIS

XSI

```
#include <grp.h>

struct group *getgrent(void);
```

DESCRIPTION

Refer to *endgrent( )*.

38620 **NAME**

38621 getgrgid, getgrgid_r — get group database entry for a group ID

38622 **SYNOPSIS**

38623 ```
#include <grp.h>
```

38624 ```
struct group *getgrgid(gid_t gid);
```

38625 ```
int getgrgid_r(gid_t gid, struct group *grp, char *buffer,
```

38626 ```
    size_t bufsize, struct group **result);
```

38627 **DESCRIPTION**

38628 The *getgrgid*( ) function shall search the group database for an entry with a matching *gid*.

38629 The *getgrgid*( ) function need not be thread-safe.

38630 Applications wishing to check for error situations should set *errno* to 0 before calling *getgrgid*( ).
38631 If *getgrgid*( ) returns a null pointer and *errno* is set to non-zero, an error occurred.

38632 The *getgrgid_r*( ) function shall update the **group** structure pointed to by *grp* and store a pointer
38633 to that structure at the location pointed to by *result*. The structure shall contain an entry from
38634 the group database with a matching *gid*. Storage referenced by the group structure is allocated
38635 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to
38636 *sysconf* (_SC_GETGR_R_SIZE_MAX) returns either –1 without changing *errno* or an initial value
38637 suggested for the size of this buffer. A null pointer shall be returned at the location pointed to
38638 by *result* on error or if the requested entry is not found.

38639 **RETURN VALUE**

38640 Upon successful completion, *getgrgid*( ) shall return a pointer to a **struct group** with the structure
38641 defined in **<grp.h>** with a matching entry if one is found. The *getgrgid*( ) function shall return a
38642 null pointer if either the requested entry was not found, or an error occurred. If the requested
38643 entry was not found, *errno* shall not be changed. On error, *errno* shall be set to indicate the error.

38644 The application shall not modify the structure to which the return value points, nor any storage
38645 areas pointed to by pointers within the structure. The returned pointer, and pointers within the
38646 structure, might be invalidated or the structure or the storage areas might be overwritten by a
38647 subsequent call to *getgrent*( ), *getgrgid*( ), or *getgrnam*( ). The returned pointer, and pointers
38648 within the structure, might also be invalidated if the calling thread is terminated.

38649 If successful, the *getgrgid_r*( ) function shall return zero; otherwise, an error number shall be
38650 returned to indicate the error.

38651 **ERRORS**

38652 The *getgrgid*( ) and *getgrgid_r*( ) functions may fail if:

38653 [EIO] An I/O error has occurred.

38654 [EINTR] A signal was caught during *getgrgid*( ).

38655 [EMFILE] All file descriptors available to the process are currently open.

38656 [ENFILE] The maximum allowable number of files is currently open in the system.

38657 The *getgrgid_r*( ) function shall fail if:

38658 [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be
38659 referenced by the resulting **group** structure.

EXAMPLES

Note that *sysconf*(_SC_GETGR_R_SIZE_MAX) may return –1 if there is no hard limit on the size of the buffer needed to store all the groups returned. This example shows how an application can allocate a buffer of sufficient size to work with *getgrid_r*( ).

```
long int initlen = sysconf(_SC_GETGR_R_SIZE_MAX);
size_t len;
if (initlen == -1)
    /* Default initial length. */
    len = 1024;
else
    len = (size_t) initlen;
struct group result;
struct group *resultp;
char *buffer = malloc(len);
if (buffer == NULL)
    ...handle error...
int e;
while ((e = getgrgid_r(42, &result, buffer, len, &resultp)) == ERANGE)
    {
    size_t newlen = 2 * len;
    if (newlen < len)
        ...handle error...
    len = newlen;
    char *newbuffer = realloc(buffer, len);
    if (newbuffer == NULL)
        ...handle error...
    buffer = newbuffer;
    }
if (e != 0)
    ...handle error...
free (buffer);
```

Finding an Entry in the Group Database

The following example uses *getgrgid*( ) to search the group database for a group ID that was previously stored in a **stat** structure, then prints out the group name if it is found. If the group is not found, the program prints the numeric value of the group for the entry.

```
#include <sys/types.h>
#include <grp.h>
#include <stdio.h>
...
struct stat statbuf;
struct group *grp;
...
if ((grp = getgrgid(statbuf.st_gid)) != NULL)
    printf(" %-8.8s", grp->gr_name);
else
    printf(" %-8d", statbuf.st_gid);
...
```


APPLICATION USAGE

The *getgrgid_r*( ) function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

Portable applications should take into account that it is usual for an implementation to return –1 from *sysconf*( ) indicating that there is no maximum for _SC_GETGR_R_SIZE_MAX.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endgrent( ), *getgrnam*( ), *sysconf*( )

XBD **<grp.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 1. Derived from System V Release 2.0.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

The *getgrgid_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *getgrgid*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *getgrgid_r*( ) function is marked as part of the Thread-Safe Functions option.

The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION describing matching the *gid*.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the buffer from *bufsize* characters to bytes.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-166 is applied.

The *getgrgid_r*( ) function is moved from the Thread-Safe Functions option to the Base.

A minor addition is made to the EXAMPLES section, reminding the application developer to free memory allocated as if by *malloc*( ).

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0241 [75] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0160 [808], XSH/TC2-2008/0161 [808], XSH/TC2-2008/0162 [656], and XSH/TC2-2008/0163 [808] are applied.

Issue 8

Austin Group Defect 398 is applied, changing the [ERANGE] error from ``may fail'' to ``shall fail''.

Austin Group Defect 1570 is applied, removing extra spacing in `"=="`.

38758 **NAME**

38759 getgrnam, getgrnam_r — search group database for a name

38760 **SYNOPSIS**

38761 ```
#include <grp.h>

struct group *getgrnam(const char *name);
int getgrnam_r(const char *name, struct group *grp, char *buffer,
    size_t bufsize, struct group **result);
```

38765 **DESCRIPTION**

38766 The *getgrnam*( ) function shall search the group database for an entry with a matching *name*.

38767 The *getgrnam*( ) function need not be thread-safe.

38768 Applications wishing to check for error situations should set *errno* to 0 before calling *getgrnam*( ).
38769 If *getgrnam*( ) returns a null pointer and *errno* is set to non-zero, an error occurred.

38770 The *getgrnam_r*( ) function shall update the **group** structure pointed to by *grp* and store a pointer
38771 to that structure at the location pointed to by *result*. The structure shall contain an entry from
38772 the group database with a matching *name*. Storage referenced by the **group** structure is allocated
38773 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to
38774 *sysconf* (_SC_GETGR_R_SIZE_MAX) returns either –1 without changing *errno* or an initial value
38775 suggested for the size of this buffer. A null pointer is returned at the location pointed to by *result*
38776 on error or if the requested entry is not found.

38777 **RETURN VALUE**

38778 The *getgrnam*( ) function shall return a pointer to a **struct group** with the structure defined in
38779 **<grp.h>** with a matching entry if one is found. The *getgrnam*( ) function shall return a null
38780 pointer if either the requested entry was not found, or an error occurred. If the requested entry
38781 was not found, *errno* shall not be changed. On error, *errno* shall be set to indicate the error.

38782 The application shall not modify the structure to which the return value points, nor any storage
38783 areas pointed to by pointers within the structure. The returned pointer, and pointers within the
38784 structure, might be invalidated or the structure or the storage areas might be overwritten by a
38785 subsequent call to *getgrent*( ), *getgrgid*( ), or *getgrnam*( ). The returned pointer, and pointers
38786 within the structure, might also be invalidated if the calling thread is terminated.

38787 The *getgrnam_r*( ) function shall return zero on success or if the requested entry was not found
38788 and no error has occurred. If any error has occurred, an error number shall be returned to
38789 indicate the error.

38790 **ERRORS**

38791 The *getgrnam*( ) and *getgrnam_r*( ) functions may fail if:

38792 [EIO] An I/O error has occurred.

38793 [EINTR] A signal was caught during *getgrnam*( ).

38794 [EMFILE] All file descriptors available to the process are currently open.

38795 [ENFILE] The maximum allowable number of files is currently open in the system.

38796 The *getgrnam_r*( ) function shall fail if:

38797 [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be
38798 referenced by the resulting **group** structure.

  
```

EXAMPLES

Note that *sysconf*(_SC_GETGR_R_SIZE_MAX) may return –1 if there is no hard limit on the size of the buffer needed to store all the groups returned. This example shows how an application can allocate a buffer of sufficient size to work with *getgrnam_r*( ).

```
long int initlen = sysconf(_SC_GETGR_R_SIZE_MAX);
size_t len;
if (initlen == -1)
    /* Default initial length. */
    len = 1024;
else
    len = (size_t) initlen;
struct group result;
struct group *resultp;
char *buffer = malloc(len);
if (buffer == NULL)
    ...handle error...
int e;
while ((e = getgrnam_r("somegroup", &result, buffer, len, &resultp))
        == ERANGE)
    {
    size_t newlen = 2 * len;
    if (newlen < len)
        ...handle error...
    len = newlen;
    char *newbuffer = realloc(buffer, len);
    if (newbuffer == NULL)
        ...handle error...
    buffer = newbuffer;
    }
if (e != 0)
    ...handle error...
free (buffer);
```

APPLICATION USAGE

The *getgrnam_r*( ) function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

Portable applications should take into account that it is usual for an implementation to return –1 from *sysconf*( ) indicating that there is no maximum for _SC_GETGR_R_SIZE_MAX.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endgrent( ), *getgrgid*( ), *sysconf*( )

XBD **<grp.h>**, **<sys/types.h>**

38843 **CHANGE HISTORY**

38844 First released in Issue 1. Derived from System V Release 2.0.

38845 **Issue 5**

38846 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
38847 VALUE section.

38848 The *getgrnam_r*( ) function is included for alignment with the POSIX Threads Extension.

38849 A note indicating that the *getgrnam*( ) function need not be reentrant is added to the
38850 DESCRIPTION.

38851 **Issue 6**

38852 The *getgrnam_r*( ) function is marked as part of the Thread-Safe Functions option.

38853 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

38854 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

38855 The following new requirements on POSIX implementations derive from alignment with the
38856 Single UNIX Specification:

38857 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
38858 required for conforming implementations of previous POSIX specifications, it was not
38859 required for UNIX applications.

38860 • In the RETURN VALUE section, the requirement to set *errno* on error is added.

38861 • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

38862 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
38863 its avoidance of possibly using a static data area.

38864 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the
38865 buffer from *bufsize* characters to bytes.

38866 **Issue 7**

38867 Austin Group Interpretation 1003.1-2001 #081 is applied, clarifying the RETURN VALUE section.

38868 Austin Group Interpretation 1003.1-2001 #156 is applied.

38869 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

38870 SD5-XSH-ERN-166 is applied.

38871 The *getgrnam_r*( ) function is moved from the Thread-Safe Functions option to the Base.

38872 A minor addition is made to the EXAMPLES section, reminding the application developer to
38873 free memory allocated as if by *malloc*( ).

38874 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0242 [75] is applied.

38875 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0164 [808], XSH/TC2-2008/0165 [808],
38876 XSH/TC2-2008/0166 [656], and XSH/TC2-2008/0167 [808] are applied.

38877 **Issue 8**

38878 Austin Group Defect 398 is applied, changing the [ERANGE] error from ``may fail'' to ``shall
38879 fail''.

38880 Austin Group Defect 1570 is applied, removing extra spacing in `"=="`.

NAME

getgroups — get supplementary group IDs

SYNOPSIS

```
#include <unistd.h>

int getgroups(int gidsetsize, gid_t grouplist[]);
```

DESCRIPTION

The *getgroups*( ) function shall fill in the array *grouplist* with the current supplementary group IDs of the calling process. It is implementation-defined whether *getgroups*( ) also returns the effective group ID in the *grouplist* array.

The *gidsetsize* argument specifies the number of elements in the array *grouplist*. The actual number of group IDs stored in the array shall be returned. The values of array entries with indices greater than or equal to the value returned are undefined.

If *gidsetsize* is 0, *getgroups*( ) shall return the number of group IDs that it would otherwise return without modifying the array pointed to by *grouplist*.

If the effective group ID of the process is returned with the supplementary group IDs, the value returned shall always be greater than or equal to one and less than or equal to the value of {NGROUPS_MAX}+1.

RETURN VALUE

Upon successful completion, the number of supplementary group IDs shall be returned. A return value of –1 indicates failure and *errno* shall be set to indicate the error.

ERRORS

The *getgroups*( ) function shall fail if:

[EINVAL] The *gidsetsize* argument is non-zero and less than the number of group IDs that would have been returned.

EXAMPLES

Getting the Supplementary Group IDs of the Calling Process

The following example places the current supplementary group IDs of the calling process into the *group* array.

```
#include <sys/types.h>
#include <unistd.h>
...
gid_t *group;
int ngroups;
long ngroups_max;

ngroups_max = sysconf(_SC_NGROUPS_MAX) + 1;
group = (gid_t *)malloc(ngroups_max *sizeof(gid_t));

ngroups = getgroups(ngroups_max, group);
```

APPLICATION USAGE

None.

RATIONALE

The related function *setgroups*( ) is a privileged operation and therefore is not covered by this volume of POSIX.1-2024.

As implied by the definition of supplementary groups, the effective group ID may appear in the

array returned by *getgroups*( ) or it may be returned only by *getegid*( ). Duplication may exist, but the application needs to call *getegid*( ) to be sure of getting all of the information. Various implementation variations and administrative sequences cause the set of groups appearing in the result of *getgroups*( ) to vary in order and as to whether the effective group ID is included, even when the set of groups is the same (in the mathematical sense of ``set''). (The history of a process and its parents could affect the details of the result.)

Application developers should note that {NGROUPS_MAX} is not necessarily a constant on all implementations.

FUTURE DIRECTIONS

None.

SEE ALSO

getegid( ), *setgid*( )

XBD **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- A return value of 0 is not permitted, because {NGROUPS_MAX} cannot be 0. This is a FIPS requirement.

The following changes were made to align with the IEEE P1003.1a draft standard:

- An explanation is added that the effective group ID may be included in the supplementary group list.

Issue 8

Austin Group Defect 1400 is applied, changing the EXAMPLES section.

38956 **NAME**

38957 gethostent — network host database functions

38958 **SYNOPSIS**

38959 `#include <netdb.h>`

38960 `struct hostent *gethostent(void);`

38961 **DESCRIPTION**

38962 Refer to *endhostent( )*.

38963 **NAME**

38964 gethostid — get an identifier for the current host

38965 **SYNOPSIS**

38966 XSI
```
#include <unistd.h>
```
38967
```
long gethostid(void);
```

38968 **DESCRIPTION**

38969 The *gethostid*( ) function shall retrieve a 32-bit identifier for the current host.

38970 **RETURN VALUE**

38971 Upon successful completion, *gethostid*( ) shall return an identifier for the current host.

38972 **ERRORS**

38973 No errors are defined.

38974 **EXAMPLES**

38975 None.

38976 **APPLICATION USAGE**

38977 This volume of POSIX.1-2024 does not define the domain in which the return value is unique.

38978 **RATIONALE**

38979 None.

38980 **FUTURE DIRECTIONS**

38981 None.

38982 **SEE ALSO**

38983 *initstate*( )

38984 XBD **<unistd.h>**

38985 **CHANGE HISTORY**

38986 First released in Issue 4, Version 2.

38987 **Issue 5**

38988 Moved from X/OPEN UNIX extension to BASE.

NAME

gethostname — get name of current host

SYNOPSIS

```
#include <unistd.h>

int gethostname(char *name, size_t namelen);
```

DESCRIPTION

The *gethostname*( ) function shall return the standard host name for the current machine. The *namelen* argument shall specify the size of the array pointed to by the *name* argument. The returned name shall be null-terminated, except that if *namelen* is an insufficient length to hold the host name, then the returned name shall be truncated and it is unspecified whether the returned name is null-terminated.

Host names are limited to {HOST_NAME_MAX} bytes.

RETURN VALUE

Upon successful completion, 0 shall be returned; otherwise, –1 shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

gethostid( ), *uname*( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The Open Group Base Resolution bwg2001-008 is applied, changing the *namelen* parameter from **socklen_t** to **size_t**.

39020 **NAME**

39021 getline — read a delimited record from *stream*

39022 **SYNOPSIS**

39023 CX
```
#include <stdio.h>

ssize_t getline(char **restrict lineptr, size_t *restrict n,
    FILE *restrict stream);
```

39026 **DESCRIPTION**

39027 Refer to *getdelim( )*.

NAME

getlocalename_l — get a locale name from a locale object

SYNOPSIS

CX
```
#include <locale.h>

const char *getlocalename_l(int category, locale_t locobj);
```

DESCRIPTION

If *category* is not LC_ALL, the *getlocalename_l*( ) function shall return the locale name for the given locale category of the locale object *locobj*, or of the global locale if *locobj* is the special locale object LC_GLOBAL_LOCALE.

If *category* is LC_ALL, the *getlocalename_l*( ) function shall return a string that encodes the locale settings for all locale categories of the locale object *locobj*, or of the global locale if *locobj* is the special locale object LC_GLOBAL_LOCALE, in the same form as is returned by *setlocale*( ). The string returned is such that a subsequent call to *setlocale*( ), from the same process, with a pointer to that string as *locale* and the LC_ALL *category* shall set the global locale to the same locale for each category as was present in the queried object.

If the value of the *category* argument is neither LC_ALL nor a supported locale category value (see *setlocale*( )), *getlocalename_l*( ) shall fail.

The behavior is undefined if the *locobj* argument is neither the special locale object LC_GLOBAL_LOCALE nor a valid locale object handle.

RETURN VALUE

Upon successful completion, *getlocalename_l*( ) shall return a pointer to a string; otherwise, a null pointer shall be returned.

If *locobj* is LC_GLOBAL_LOCALE, the returned string pointer might be invalidated or the string content might be overwritten by a subsequent call in the same thread to *getlocalename_l*( ) with LC_GLOBAL_LOCALE; the returned string pointer might also be invalidated if the calling thread is terminated. Otherwise, the returned string pointer and content shall remain valid until the locale object *locobj* is used in a call to *freelocale*( ) or as the *base* argument in a successful call to *newlocale*( ).

ERRORS

No errors are defined.

EXAMPLES

Determining the locale name for a category of the current locale

The following example shows how to obtain the locale name for the LC_NUMERIC category of the current thread-local locale, or of the global locale if no thread-local locale is in use.

```
#include <locale.h>
...
const char *name;
locale_t loc = uselocale(NULL);
name = getlocalename_l(LC_NUMERIC, loc);
```

APPLICATION USAGE

In addition to the caveats regarding validity of the returned string pointer in RETURN VALUE, the content of the string returned when *category* is LC_ALL is only required to be valid for the life of the process, so is not intended for storage or sharing between processes. As the internal format of the string is implementation-specific, there is nothing preventing a subsequent run of

an application from being presented a different format, for example if the implementation is updated.

RATIONALE

Historical versions of *getlocalename_l*( ) did not handle the special locale object LC_GLOBAL_LOCALE, requiring that applications used *setlocale*(*category*, NULL) to query the global locale if *uselocale*(NULL) returned LC_GLOBAL_LOCALE. However, since *setlocale*( ) is not required to be thread-safe (even when the only concurrent calls are ones that query the locale), this method was problematic for multi-threaded processes. This standard requires that *getlocalename_l*(*category*, LC_GLOBAL_LOCALE) queries the global locale in a thread-safe manner, for example by returning a pointer to a thread-local internal buffer instead of a process-wide internal buffer.

FUTURE DIRECTIONS

None.

SEE ALSO

freelocale( ), *newlocale*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<locale.h>**

CHANGE HISTORY

First released in Issue 8.

NAME

getlogin, getlogin_r — get login name

SYNOPSIS

```
#include <unistd.h>

char *getlogin(void);
int getlogin_r(char *name, size_t namesize);
```

DESCRIPTION

The *getlogin*( ) function shall return a pointer to a string containing the user name associated by the login activity with the controlling terminal of the current process. If *getlogin*( ) returns a non-null pointer, then that pointer points to the name that the user logged in under, even if there are several login names with the same user ID.

The *getlogin*( ) function need not be thread-safe.

The *getlogin_r*( ) function shall put the name associated by the login activity with the controlling terminal of the current process in the character array pointed to by *name*. The array is *namesize* characters long and should have space for the name and the terminating null character. The maximum size of the login name is {LOGIN_NAME_MAX}.

If *getlogin_r*( ) is successful, *name* points to the name the user used at login, even if there are several login names with the same user ID.

The *getlogin*( ) and *getlogin_r*( ) functions may make use of file descriptors 0, 1, and 2 to find the controlling terminal of the current process, examining each in turn until the terminal is found. If in this case none of these three file descriptors is open to the controlling terminal, these functions may fail. The method used to find the terminal associated with a file descriptor may depend on the file descriptor being open to the actual terminal device, not **/dev/tty**.

RETURN VALUE

Upon successful completion, *getlogin*( ) shall return a pointer to the login name or a null pointer if the user's login name cannot be found. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

The application shall not modify the string returned. The returned pointer might be invalidated or the string content might be overwritten by a subsequent call to *getlogin*( ). The returned pointer and the string content might also be invalidated if the calling thread is terminated.

If successful, the *getlogin_r*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

[EMFILE] All file descriptors available to the process are currently open.

[ENFILE] The maximum allowable number of files is currently open in the system.

[ENOTTY] None of the file descriptors 0, 1, or 2 is open to the controlling terminal of the current process.

[ENXIO] The calling process has no controlling terminal.

The *getlogin_r*( ) function shall fail if:

[ERANGE] The value of *namesize* is smaller than the length of the string to be returned including the terminating null character.

EXAMPLES

Getting the User Login Name

The following example calls the *getlogin*( ) function to obtain the name of the user associated with the calling process, and passes this information to the *getpwnam*( ) function to get the associated user database information.

```
#include <unistd.h>
#include <sys/types.h>
#include <pwd.h>
#include <stdio.h>
...
char *lgn;
struct passwd *pw;
...
if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
    fprintf(stderr, "Get of user information failed.\n"); exit(1);
    }
```

APPLICATION USAGE

Three names associated with the current process can be determined: *getpwuid*(*geteuid*( )) shall return the name associated with the effective user ID of the process; *getlogin*( ) shall return the name associated with the current login activity; and *getpwuid*(*getuid*( )) shall return the name associated with the real user ID of the process.

The *getlogin_r*( ) function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

The *getlogin*( ) function returns a pointer to the user's login name. The same user ID may be shared by several login names. If it is desired to get the user database entry that is used during login, the result of *getlogin*( ) should be used to provide the argument to the *getpwnam*( ) function. (This might be used to determine the user's login shell, particularly where a single user has multiple login shells with distinct login names, but the same user ID.)

The information provided by the *cuserid*( ) function, which was originally defined in the POSIX.1-1988 standard and subsequently removed, can be obtained by the following:

```
getpwuid(geteuid())
```

while the information provided by historical implementations of *cuserid*( ) can be obtained by:

```
getpwuid(getuid())
```

The thread-safe version of this function places the user name in a user-supplied buffer and returns a non-zero value if it fails. The non-thread-safe version may return the name in a static data area that may be overwritten by each call.

FUTURE DIRECTIONS

None.

SEE ALSO

getpwnam( ), *getpwuid*( ), *geteuid*( ), *getuid*( )

XBD **<limits.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from System V Release 2.0.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

The *getlogin_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *getlogin*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *getlogin_r*( ) function is marked as part of the Thread-Safe Functions option.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

The *getlogin_r*( ) function is moved from the Thread-Safe Functions option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0243 [172], XSH/TC1-2008/0244 [75], and XSH/TC1-2008/0245 [172] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0168 [656] is applied.

Issue 8

Austin Group Defect 398 is applied, changing the [ERANGE] error from ``may fail'' to ``shall fail''.

NAME

getnameinfo — get name information

SYNOPSIS

```
#include <sys/socket.h>
#include <netdb.h>

int getnameinfo(const struct sockaddr *restrict sa, socklen_t salen,
    char *restrict node, socklen_t nodelen, char *restrict service,
    socklen_t servicelen, int flags);
```

DESCRIPTION

The *getnameinfo*( ) function shall translate a socket address to a node name and service location, all of which are defined as in *freeaddrinfo*( ).

The *sa* argument points to a socket address structure to be translated. The *salen* argument contains the length of the address pointed to by *sa*.

IP6 If the socket address structure contains an IPv4-mapped IPv6 address or an IPv4-compatible IPv6 address, the implementation shall extract the embedded IPv4 address and lookup the node name for that IPv4 address.

If the address is the IPv6 unspecified address (`"::"`), a lookup shall not be performed and the behavior shall be the same as when the node's name cannot be located.

If the *node* argument is non-NULL and the *nodelen* argument is non-zero, then the *node* argument points to a buffer able to contain up to *nodelen* bytes that receives the node name as a null-terminated string. If the *node* argument is NULL or the *nodelen* argument is zero, the node name shall not be returned. If the node's name cannot be located, the numeric form of the address contained in the socket address structure pointed to by the *sa* argument is returned instead of its name.

If the *service* argument is non-NULL and the *servicelen* argument is non-zero, then the *service* argument points to a buffer able to contain up to *servicelen* bytes that receives the service name as a null-terminated string. If the *service* argument is NULL or the *servicelen* argument is zero, the service name shall not be returned. If the service's name cannot be located, the numeric form of the service address (for example, its port number) shall be returned instead of its name.

The *flags* argument is a flag that changes the default actions of the function. By default the fully-qualified domain name (FQDN) for the host shall be returned, but:

- If the flag bit NI_NOFQDN is set, only the node name portion of the FQDN shall be returned for local hosts.
- If the flag bit NI_NUMERICHOST is set, the numeric form of the address contained in the socket address structure pointed to by the *sa* argument shall be returned instead of its name.
- If the flag bit NI_NAMEREQD is set, an error shall be returned if the host's name cannot be located.
- If the flag bit NI_NUMERICSERV is set, the numeric form of the service address shall be returned (for example, its port number) instead of its name.
- If the flag bit NI_NUMERICSCOPE is set, the numeric form of the scope identifier shall be returned (for example, interface index) instead of its name. This flag shall be ignored if the *sa* argument is not an IPv6 address.

39244 • If the flag bit NI_DGRAM is set, this indicates that the service is a datagram service
39245 (SOCK_DGRAM). The default behavior shall assume that the service is a stream service
39246 (SOCK_STREAM).

39247 **Notes:**

39248 1. The two NI_NUMERICxxx flags are required to support the –**n** flag that many
39249 commands provide.

39250 2. The NI_DGRAM flag is required for the few AF_INET and AF_INET6 port numbers (for
39251 example, [512,514]) that represent different services for UDP and TCP.

39252 The *getnameinfo*( ) function shall be thread-safe.

39253 **RETURN VALUE**

39254 A zero return value for *getnameinfo*( ) indicates successful completion; a non-zero return value
39255 indicates failure. The possible values for the failures are listed in the ERRORS section.

39256 Upon successful completion, *getnameinfo*( ) shall return the *node* and *service* names, if requested,
39257 in the buffers provided. The returned names are always null-terminated strings.

39258 **ERRORS**

39259 The *getnameinfo*( ) function shall fail and return the corresponding value if:

39260 [EAI_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

39261 [EAI_BADFLAGS]
39262 The *flags* had an invalid value.

39263 [EAI_FAIL] A non-recoverable error occurred.

39264 [EAI_FAMILY] The address family was not recognized or the address length was invalid for
39265 the specified family.

39266 [EAI_MEMORY] There was a memory allocation failure.

39267 [EAI_NONAME] The name does not resolve for the supplied parameters.

39268 NI_NAMEREQD is set and the host's name cannot be located, or both
39269 *nodename* and *servname* were null.

39270 [EAI_OVERFLOW]
39271 An argument buffer overflowed. The buffer pointed to by the *node* argument
39272 or the *service* argument was too small.

39273 [EAI_SYSTEM] A system error occurred. The error code can be found in *errno*.

39274 **EXAMPLES**

39275 None.

39276 **APPLICATION USAGE**

39277 If the returned values are to be used as part of any further name resolution (for example, passed
39278 to *getaddrinfo*( )), applications should provide buffers large enough to store any result possible on
39279 the system.

39280 Given the IPv4-mapped IPv6 address `"::ffff:1.2.3.4"`, the implementation performs a
39281 lookup as if the socket address structure contains the IPv4 address `"1.2.3.4"`.

39282 The IPv6 unspecified address (`"::"`) and the IPv6 loopback address (`"::1"`) are not
39283 IPv4-compatible addresses.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endservent( ), *freeaddrinfo*( ), *gai_strerror*( ), *inet_ntop*( ), *socket*( )

XBD **<netdb.h>**, **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The **restrict** keyword is added to the *getnameinfo*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/23 is applied, making various changes in the SYNOPSIS and DESCRIPTION for alignment with IPv6.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/24 is applied, adding the [EAI_OVERFLOW] error to the ERRORS section.

Issue 7

SD5-XSH-ERN-127 is applied, clarifying the behavior if the address is the IPv6 unspecified address.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0246 [284] and XSH/TC1-2008/0247 [285] are applied.

39304 **NAME**

39305 getnetbyaddr, getnetbyname, getnetent — network database functions

39306 **SYNOPSIS**

39307 `#include <netdb.h>`

39308 `struct netent *getnetbyaddr(uint32_t net, int type);`
39309 `struct netent *getnetbyname(const char *name);`
39310 `struct netent *getnetent(void);`

39311 **DESCRIPTION**

39312 Refer to *endnetent( )*.

NAME

getopt, optarg, opterr, optind, optopt — command option parsing

SYNOPSIS

```
#include <unistd.h>

int getopt(int argc, char * const argv[], const char *optstring);
extern char *optarg;
extern int opterr, optind, optopt;
```

DESCRIPTION

The *getopt*( ) function is a command-line parser that shall follow Utility Syntax Guidelines 3, 4, 5, 6, 7, 9, and 10 in XBD Section 12.2 (on page 215).

The parameters *argc* and *argv* are the argument count and argument array as passed to *main*( ) (see *exec*( )). The argument *optstring* is a string of recognized option characters; if a character is followed by a <colon>, the option takes an argument. All option characters allowed by Utility Syntax Guideline 3 are allowed in *optstring*. The *optstring* argument can optionally start with a <plus-sign> (`'+'`), which shall have no effect on behavior in a conforming environment. If a <plus-sign> occurs anywhere besides the first character of *optstring*, the behavior is unspecified. The implementation may accept other characters as an extension.

The variable *optind* is the index of the next element of the *argv*[ ] vector to be processed. It shall be initialized to 1 by the system, and *getopt*( ) shall update it when it finishes with each element of *argv*[ ]. If the application sets *optind* to zero before calling *getopt*( ), the behavior is unspecified. When an element of *argv*[ ] contains multiple option characters, it is unspecified how *getopt*( ) determines which options have already been processed.

The *getopt*( ) function shall return the next option character (if one is found) from *argv* that matches a character in *optstring* (excluding an optional leading <plus-sign>), if there is one that matches. If the option takes an argument, *getopt*( ) shall set the variable *optarg* to point to the option-argument as follows:

1. If the option was the last character in the string pointed to by an element of *argv*, then *optarg* shall contain the next element of *argv*, and *optind* shall be incremented by 2. If the resulting value of *optind* is greater than *argc*, this indicates a missing option-argument, and *getopt*( ) shall return an error indication.
2. Otherwise, *optarg* shall point to the string following the option character in that element of *argv*, and *optind* shall be incremented by 1.

If, when *getopt*( ) is called, any of the following is true:

argv[optind]	is a null pointer
**argv*[optind]	is not the character `'-'`
argv[optind]	points to the string `"-"`

getopt( ) shall return –1 without changing *optind*. If:

argv[optind]	points to the string `"--"`

getopt( ) shall return –1 after incrementing *optind*.

If *getopt*( ) encounters a <colon> as an option character, or an option character that is not contained in *optstring* after an optional leading <plus-sign>, it shall return the <question-mark> (`'?'`) character. If it detects a missing option-argument, it shall return the <colon> character (`':'`) if the first character of *optstring* after an optional <plus-sign> was a <colon>, or a <question-mark> character (`'?'`) otherwise. In either case, *getopt*( ) shall set the variable *optopt* to the option character that caused the error. If the application has not set the variable *opterr* to 0,

and the first character of *optstring* after an optional <plus-sign> is not a <colon>, *getopt*( ) shall also print a diagnostic message to *stderr* in the format specified for the *getopts* utility, unless the *stderr* stream has wide orientation, in which case the behavior is undefined.

The *getopt*( ) function need not be thread-safe.

RETURN VALUE

The *getopt*( ) function shall return the next option character specified on the command line.

A <colon> (`':'`) shall be returned if *getopt*( ) detects a missing argument and the first character of *optstring* after an optional <plus-sign> was a <colon> (`':'`).

A <question-mark> (`'?'`) shall be returned if *getopt*( ) encounters a <colon> as an option character, encounters an option character not in *optstring*, or detects a missing argument and the first character of *optstring* after an optional <plus-sign> was not a <colon> (`':'`).

Otherwise, *getopt*( ) shall return –1 when all command line options are parsed.

ERRORS

If the application has not set the variable *opterr* to 0, the first character of *optstring* is not a <colon>, and a write error occurs while *getopt*( ) is printing a diagnostic message to *stderr*, then the error indicator for *stderr* shall be set; but *getopt*( ) shall still succeed and the value of *errno* after *getopt*( ) is unspecified.

EXAMPLES

Parsing Command Line Options

The following code fragment shows how you might process the arguments for a utility that can take the mutually-exclusive options *a* and *b* and the options *f* and *o*, both of which require arguments:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int
main(int argc, char *argv[ ])
{
    int c;
    int bflg = 0, aflg = 0, errflg = 0;
    char *ifile;
    char *ofile;
    . . .
    while ((c = getopt(argc, argv, ":abf:o:")) != -1) {
        switch(c) {
        case 'a':
            if (bflg)
                errflg++;
            else
                aflg++;
            break;
        case 'b':
            if (aflg)
                errflg++;
            else
                bflg++;
```



```
                break;
            case 'f':
                ifile = optarg;
                break;
            case 'o':
                ofile = optarg;
                break;
            case ':':       /* -f or -o without operand */
                fprintf(stderr,
                    "Option -%c requires an operand\n", optopt);
                errflg++;
                break;
            case '?':
                fprintf(stderr,
                    "Unrecognized option: '-%c'\n", optopt);
                errflg++;
            }
        }
        if (errflg) {
            fprintf(stderr, "usage: . . . ");
            exit(2);
        }
        for ( ; optind < argc; optind++) {
            if (access(argv[optind], R_OK)) {
                . . .
            }
        }
    }
```

This code accepts any of the following as equivalent:

```
cmd -ao arg path path
cmd -a -o arg path path
cmd -o arg -a path path
cmd -a -o arg -- path path
cmd -a -oarg path path
cmd -aoarg path path
```

Selecting Options from the Command Line

The following example selects the type of database routines the user wants to use based on the *Options* argument.

```
#include <unistd.h>
#include <string.h>
...
const char *Options = "hdbtl";
...
int dbtype, c;
char *st;
...
dbtype = 0;
while ((c = getopt(argc, argv, Options)) != -1) {
    if ((st = strchr(Options, c)) != NULL) {
```



```
                dbtype = st - Options;
                break;
            }
        }
```

APPLICATION USAGE

The *getopt*( ) function is only required to support option characters included in Utility Syntax Guideline 3. Many historical implementations of *getopt*( ) support other characters as options. This is an allowed extension, but applications that use extensions are not maximally portable. Note that support for multi-byte option characters is only possible when such characters can be represented as type **int**.

Applications which use wide-character output functions with *stderr* should ensure that any calls to *getopt*( ) do not write to *stderr*, either by setting *opterr* to 0 or by ensuring the first character of *optstring* is always a <colon>.

While *ferror*(*stderr*) may be used to detect failures to write a diagnostic to *stderr* when *getopt*( ) returns `'?'`, the value of *errno* is unspecified in such a condition. Applications desiring more control over handling write failures should set *opterr* to 0 and independently perform output to *stderr*, rather than relying on *getopt*( ) to do the output.

RATIONALE

The *optopt* variable represents historical practice and allows the application to obtain the identity of the invalid option.

The description has been written to make it clear that *getopt*( ), like the *getopts* utility, deals with option-arguments whether separated from the option by <blank> characters or not. Note that the requirements on *getopt*( ) and *getopts* are more stringent than the Utility Syntax Guidelines.

The *getopt*( ) function shall return –1, rather than EOF, so that **<stdio.h>** is not required.

The special significance of a <colon> as the first character of *optstring* makes *getopt*( ) consistent with the *getopts* utility. It allows an application to make a distinction between a missing argument and an incorrect option letter without having to examine the option letter. It is true that a missing argument can only be detected in one case, but that is a case that has to be considered.

In some non-conforming environments, the use of a leading <plus-sign> in *optstring* forces *getopt*( ) to behave in a conforming way, when it would otherwise have non-conforming behavior. Its use has been standardized to allow applications to be written that can guarantee behavior consistent with this specification even in an otherwise non-conforming environment. If both <plus-sign> and <colon> are used at the beginning of *optstring*, the <plus-sign> must be first.

Note that the use of a leading <plus-sign> in *optstring* is only standardized for *getopt*( ). Use of a <plus-sign> is intentionally left unspecified for the *getopts* utility, where historical implementations did not require a leading <plus-sign> for conforming behavior, and because some historical *getopts* implementations used a leading <plus-sign> for a different extension.

FUTURE DIRECTIONS

None.

SEE ALSO

exec

XBD Section 12.2 (on page 215), **<unistd.h>**

XCU *getopts*

39498 **CHANGE HISTORY**

39499 First released in Issue 1. Derived from Issue 1 of the SVID.

39500 **Issue 5**

39501 A note indicating that the *getopt*( ) function need not be reentrant is added to the DESCRIPTION.

39502 **Issue 6**

39503 IEEE PASC Interpretation 1003.2 #150 is applied.

39504 Austin Group Interpretation 1003.1-2001 #156 is applied.

39505 **Issue 7**

39506 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0248 [318], XSH/TC1-2008/0249 [460],
39507 XSH/TC1-2008/0250 [189], XSH/TC1-2008/0251 [189], XSH/TC1-2008/0252 [189], and
39508 XSH/TC1-2008/0253 [460] are applied.

39509 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0169 [608] is applied.

39510 **Issue 8**

39511 Austin Group Defect 191 is applied, allowing a leading <plus-sign> in *optstring*.

39512 Austin Group Defect 1179 is applied, adding some missing `'}'` characters at the end of the
39513 example code.

39514 Austin Group Defect 1523 is applied, clarifying the conditions under which *getopt*( ) returns –1
39515 without changing *optind*.

39516 **NAME**

39517 getpeername — get the name of the peer socket

39518 **SYNOPSIS**

39519 ```
#include <sys/socket.h>

int getpeername(int socket, struct sockaddr *restrict address,
    socklen_t *restrict address_len);
```

39522 **DESCRIPTION**

39523 The *getpeername*( ) function shall retrieve the peer address of the specified socket, store this
39524 address in the **sockaddr** structure pointed to by the *address* argument, and store the length of this
39525 address in the object pointed to by the *address_len* argument.

39526 The *address_len* argument points to a **socklen_t** object which on input specifies the length of the
39527 supplied **sockaddr** structure, and on output specifies the length of the peer address. If the actual
39528 length of the address is greater than the length of the supplied **sockaddr** structure, the stored
39529 address shall be truncated.

39530 If the protocol permits connections by unbound clients, and the peer is not bound, then the
39531 value stored in the object pointed to by *address* is unspecified.

39532 **RETURN VALUE**

39533 Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to
39534 indicate the error.

39535 **ERRORS**

39536 The *getpeername*( ) function shall fail if:

39537 [EBADF] The *socket* argument is not a valid file descriptor.

39538 [EINVAL] The socket has been shut down.

39539 [ENOTCONN] The socket is not connected or otherwise has not had the peer pre-specified.

39540 [ENOTSOCK] The *socket* argument does not refer to a socket.

39541 [EOPNOTSUPP] The operation is not supported for the socket protocol.

39542 The *getpeername*( ) function may fail if:

39543 [ENOBUFS] Insufficient resources were available in the system to complete the call.

39544 **EXAMPLES**

39545 None.

39546 **APPLICATION USAGE**

39547 For AF_UNIX sockets, it is recommended that *address* points to a buffer of length greater than
39548 `sizeof(struct sockaddr_un)` which has been initialized with null bytes. That way, even if
39549 the implementation supports the use of all bytes of *sun_path* without a terminating null byte, the
39550 larger buffer guarantees that the *sun_path* member can then be passed to other interfaces that
39551 expect a null-terminated string. If no truncation occurred based on the input value of *address_len*,
39552 it is unspecified whether the returned *address_len* will be `sizeof(struct sockaddr_un)`, or
39553 merely a value at least as large as `offsetof(struct sockaddr_un, sun_path)` plus the
39554 number of non-null bytes stored in *sun_path*.

39555 **RATIONALE**

39556 None.

  
```

39557 **FUTURE DIRECTIONS**

39558 None.

39559 **SEE ALSO**

39560 *accept*( ), *bind*( ), *getsockname*( ), *socket*( )

39561 XBD **<sys/socket.h>**

39562 **CHANGE HISTORY**

39563 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

39564 The **restrict** keyword is added to the *getpeername*( ) prototype for alignment with the
39565 ISO/IEC 9899:1999 standard.

39566 **Issue 7**

39567 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0254 [464] is applied.

39568 **Issue 8**

39569 Austin Group Defect 561 is applied, adding a paragraph about *sun_path* to APPLICATION
39570 USAGE.

39571 Austin Group Defect 1565 is applied, changing the description of *address_len*.

39572 **NAME**

39573 getpgid — get the process group ID for a process

39574 **SYNOPSIS**

39575 `#include <unistd.h>`

39576 `pid_t getpgid(pid_t pid);`

39577 **DESCRIPTION**

39578 The *getpgid*( ) function shall return the process group ID of the process whose process ID is equal
39579 to *pid*. If *pid* is equal to 0, *getpgid*( ) shall return the process group ID of the calling process.

39580 **RETURN VALUE**

39581 Upon successful completion, *getpgid*( ) shall return a process group ID. Otherwise, it shall return
39582 (**pid_t**)–1 and set *errno* to indicate the error.

39583 **ERRORS**

39584 The *getpgid*( ) function shall fail if:

39585 [EPERM] The process whose process ID is equal to *pid* is not in the same session as the
39586 calling process, and the implementation does not allow access to the process
39587 group ID of that process from the calling process.

39588 [ESRCH] There is no process with a process ID equal to *pid*.

39589 The *getpgid*( ) function may fail if:

39590 [EINVAL] The value of the *pid* argument is invalid.

39591 **EXAMPLES**

39592 None.

39593 **APPLICATION USAGE**

39594 None.

39595 **RATIONALE**

39596 None.

39597 **FUTURE DIRECTIONS**

39598 None.

39599 **SEE ALSO**

39600 *exec*, *fork*( ), *getpgrp*( ), *getpid*( ), *getsid*( ), *setpgid*( ), *setsid*( )

39601 XBD **<unistd.h>**

39602 **CHANGE HISTORY**

39603 First released in Issue 4, Version 2.

39604 **Issue 5**

39605 Moved from X/OPEN UNIX extension to BASE.

39606 **Issue 7**

39607 The *getpgid*( ) function is moved from the XSI option to the Base.

NAME

getpgrp — get the process group ID of the calling process

SYNOPSIS

```
#include <unistd.h>

pid_t getpgrp(void);
```

DESCRIPTION

The *getpgrp*( ) function shall return the process group ID of the calling process.

RETURN VALUE

The *getpgrp*( ) function shall always be successful and no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

4.3 BSD provides a *getpgrp*( ) function that returns the process group ID for a specified process. Although this function supports job control, all known job control shells always specify the calling process with this function. Thus, the simpler System V *getpgrp*( ) suffices, and the added complexity of the 4.3 BSD *getpgrp*( ) is provided by the *getpgid*( ) function.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fork*( ), *getpgid*( ), *getpid*( ), *getppid*( ), *kill*( ), *setpgid*( ), *setsid*( )

XBD **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

Issue 8

Austin Group Defect 1245 is applied, changing the RATIONALE section.

39645 **NAME**

39646 getpid — get the process ID

39647 **SYNOPSIS**

39648 `#include <unistd.h>`

39649 `pid_t getpid(void);`

39650 **DESCRIPTION**

39651 The *getpid*( ) function shall return the process ID of the calling process.

39652 **RETURN VALUE**

39653 The *getpid*( ) function shall always be successful and no return value is reserved to indicate an
39654 error.

39655 **ERRORS**

39656 No errors are defined.

39657 **EXAMPLES**

39658 None.

39659 **APPLICATION USAGE**

39660 None.

39661 **RATIONALE**

39662 None.

39663 **FUTURE DIRECTIONS**

39664 None.

39665 **SEE ALSO**

39666 *exec*, *fork*( ), *getpgrp*( ), *getppid*( ), *kill*( ), *mkdtemp*( ), *setpgid*( ), *setsid*( )

39667 XBD **<sys/types.h>**, **<unistd.h>**

39668 **CHANGE HISTORY**

39669 First released in Issue 1. Derived from Issue 1 of the SVID.

39670 **Issue 6**

39671 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

39672 The following new requirements on POSIX implementations derive from alignment with the
39673 Single UNIX Specification:

39674 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
39675 required for conforming implementations of previous POSIX specifications, it was not
39676 required for UNIX applications.

39677 **NAME**

39678 getppid — get the parent process ID

39679 **SYNOPSIS**

39680
```
#include <unistd.h>
```

39681
```
pid_t getppid(void);
```

39682 **DESCRIPTION**

39683 The *getppid*( ) function shall return the parent process ID of the calling process.

39684 **RETURN VALUE**

39685 The *getppid*( ) function shall always be successful and no return value is reserved to indicate an
39686 error.

39687 **ERRORS**

39688 No errors are defined.

39689 **EXAMPLES**

39690 None.

39691 **APPLICATION USAGE**

39692 None.

39693 **RATIONALE**

39694 None.

39695 **FUTURE DIRECTIONS**

39696 None.

39697 **SEE ALSO**

39698 *exec*, *fork*( ), *getpgid*( ), *getpgrp*( ), *getpid*( ), *kill*( ), *setpgid*( ), *setsid*( )

39699 XBD **<sys/types.h>**, **<unistd.h>**

39700 **CHANGE HISTORY**

39701 First released in Issue 1. Derived from Issue 1 of the SVID.

39702 **Issue 6**

39703 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

39704 The following new requirements on POSIX implementations derive from alignment with the
39705 Single UNIX Specification:

39706 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
39707 required for conforming implementations of previous POSIX specifications, it was not
39708 required for UNIX applications.

NAME

getpriority, setpriority — get and set the nice value

SYNOPSIS

XSI
```
#include <sys/resource.h>

int getpriority(int which, id_t who);
int setpriority(int which, id_t who, int value);
```

DESCRIPTION

The *getpriority*( ) function shall obtain the nice value of a process, process group, or user. The *setpriority*( ) function shall set the nice value of a process, process group, or user to *value*+{NZERO}.

Target processes are specified by the values of the *which* and *who* arguments. The *which* argument may be one of the following values: PRIO_PROCESS, PRIO_PGRP, or PRIO_USER, indicating that the *who* argument is to be interpreted as a process ID, a process group ID, or an effective user ID, respectively. A 0 value for the *who* argument specifies the current process, process group, or user.

The nice value set with *setpriority*( ) shall be applied to the process. If the process is multi-threaded, the nice value shall affect all system scope threads in the process.

If more than one process is specified, *getpriority*( ) shall return value {NZERO} less than the lowest nice value pertaining to any of the specified processes, and *setpriority*( ) shall set the nice values of all of the specified processes to *value*+{NZERO}.

The default nice value is {NZERO}; lower nice values shall cause more favorable scheduling. While the range of valid nice values is [0,{NZERO}*2–1], implementations may enforce more restrictive limits. If *value*+{NZERO} is less than the system's lowest supported nice value, *setpriority*( ) shall set the nice value to the lowest supported value; if *value*+{NZERO} is greater than the system's highest supported nice value, *setpriority*( ) shall set the nice value to the highest supported value.

Only a process with appropriate privileges can lower its nice value.

PS | TPS Any processes or threads using SCHED_FIFO or SCHED_RR shall be unaffected by a call to *setpriority*( ). This is not considered an error. A process which subsequently reverts to SCHED_OTHER need not have its priority affected by such a *setpriority*( ) call.

The effect of changing the nice value may vary depending on the process-scheduling algorithm in effect.

Since *getpriority*( ) can return the value –1 upon successful completion, it is necessary to set *errno* to 0 prior to a call to *getpriority*( ). If *getpriority*( ) returns the value –1, then *errno* can be checked to see if an error occurred or if the value is a legitimate nice value.

RETURN VALUE

Upon successful completion, *getpriority*( ) shall return an integer in the range –{NZERO} to {NZERO}–1. Otherwise, –1 shall be returned and *errno* set to indicate the error.

Upon successful completion, *setpriority*( ) shall return 0; otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *getpriority*( ) and *setpriority*( ) functions shall fail if:

[ESRCH] No process could be located using the *which* and *who* argument values specified.

[EINVAL] The value of the *which* argument was not recognized, or the value of the *who* argument is not a valid process ID, process group ID, or user ID.

In addition, *setpriority*( ) may fail if:

[EPERM] A process was located, but neither the real nor effective user ID of the executing process match the effective user ID of the process whose nice value is being changed.

[EACCES] A request was made to change the nice value to a lower numeric value and the current process does not have appropriate privileges.

EXAMPLES

Using getpriority( )

The following example returns the current scheduling priority for the process ID returned by the call to *getpid*( ).

```
#include <sys/resource.h>
...
int which = PRIO_PROCESS;
id_t pid;
int ret;

pid = getpid();
ret = getpriority(which, pid);
```

Using setpriority( )

The following example sets the priority for the current process ID to –20.

```
#include <sys/resource.h>
...
int which = PRIO_PROCESS;
id_t pid;
int priority = -20;
int ret;

pid = getpid();
ret = setpriority(which, pid, priority);
```

APPLICATION USAGE

The *getpriority*( ) and *setpriority*( ) functions work with an offset nice value (nice value –{NZERO}). The nice value is in the range [0,2*{NZERO} –1], while the return value for *getpriority*( ) and the third parameter for *setpriority*( ) are in the range [–{NZERO},{NZERO} –1].

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

nice( ), *sched_get_priority_max*( ), *sched_setscheduler*( )

XBD **<sys/resource.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The DESCRIPTION is reworded in terms of the nice value rather than *priority* to avoid confusion with functionality in the POSIX Realtime Extension.

39799 **NAME**

39800 getprotobyname, getprotobynumber, getprotoent — network protocol database functions

39801 **SYNOPSIS**

39802 ```
#include <netdb.h>
```

39803 ```
struct protoent *getprotobyname(const char *name);
```
39804 ```
struct protoent *getprotobynumber(int proto);
```
39805 ```
struct protoent *getprotoent(void);
```

39806 **DESCRIPTION**

39807 Refer to *endprotoent( )*.

NAME

getpwent — get user database entry

SYNOPSIS

XSI

```
#include <pwd.h>

struct passwd *getpwent(void);
```

DESCRIPTION

Refer to *endpwent( )*.

39815 **NAME**

39816 getpwnam, getpwnam_r — search user database for a name

39817 **SYNOPSIS**

39818 ```
#include <pwd.h>

struct passwd *getpwnam(const char *name);
int getpwnam_r(const char *name, struct passwd *pwd, char *buffer,
    size_t bufsize, struct passwd **result);
```

39822 **DESCRIPTION**

39823 The *getpwnam*( ) function shall search the user database for an entry with a matching *name*.

39824 The *getpwnam*( ) function need not be thread-safe.

39825 Applications wishing to check for error situations should set *errno* to 0 before calling
39826 *getpwnam*( ). If *getpwnam*( ) returns a null pointer and *errno* is non-zero, an error occurred.

39827 The *getpwnam_r*( ) function shall update the **passwd** structure pointed to by *pwd* and store a
39828 pointer to that structure at the location pointed to by *result*. The structure shall contain an entry
39829 from the user database with a matching *name*. Storage referenced by the structure is allocated
39830 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to
39831 *sysconf*(_SC_GETPW_R_SIZE_MAX) returns either –1 without changing *errno* or an initial value
39832 suggested for the size of this buffer. A null pointer shall be returned at the location pointed to
39833 by *result* on error or if the requested entry is not found.

39834 **RETURN VALUE**

39835 The *getpwnam*( ) function shall return a pointer to a **struct passwd** with the structure as defined
39836 in **<pwd.h>** with a matching entry if found. A null pointer shall be returned if the requested
39837 entry is not found, or an error occurs. If the requested entry was not found, *errno* shall not be
39838 changed. On error, *errno* shall be set to indicate the error.

39839 The application shall not modify the structure to which the return value points, nor any storage
39840 areas pointed to by pointers within the structure. The returned pointer, and pointers within the
39841 structure, might be invalidated or the structure or the storage areas might be overwritten by a
39842 subsequent call to *getpwent*( ), *getpwnam*( ), or *getpwuid*( ). The returned pointer, and pointers
39843 within the structure, might also be invalidated if the calling thread is terminated.

39844 The *getpwnam_r*( ) function shall return zero on success or if the requested entry was not found
39845 and no error has occurred. If an error has occurred, an error number shall be returned to indicate
39846 the error.

39847 **ERRORS**

39848 These functions may fail if:

39849 [EIO] An I/O error has occurred.

39850 [EINTR] A signal was caught during *getpwnam*( ).

39851 [EMFILE] All file descriptors available to the process are currently open.

39852 [ENFILE] The maximum allowable number of files is currently open in the system.

39853 The *getpwnam_r*( ) function shall fail if:

39854 [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be
39855 referenced by the resulting **passwd** structure.

  
```

EXAMPLES

Note that *sysconf*(_SC_GETPW_R_SIZE_MAX) may return –1 if there is no hard limit on the size of the buffer needed to store all the groups returned. This example shows how an application can allocate a buffer of sufficient size to work with *getpwnam_r*( ).

```
long int initlen = sysconf(_SC_GETPW_R_SIZE_MAX);
size_t len;
if (initlen == -1)
    /* Default initial length. */
    len = 1024;
else
    len = (size_t) initlen;
struct passwd result;
struct passwd *resultp;
char *buffer = malloc(len);
if (buffer == NULL)
    ...handle error...
int e;
while ((e = getpwnam_r("someuser", &result, buffer, len, &resultp))
        == ERANGE)
    {
    size_t newlen = 2 * len;
    if (newlen < len)
        ...handle error...
    len = newlen;
    char *newbuffer = realloc(buffer, len);
    if (newbuffer == NULL)
        ...handle error...
    buffer = newbuffer;
    }
if (e != 0)
    ...handle error...
free (buffer);
```

Getting an Entry for the Login Name

The following example uses the *getlogin*( ) function to return the name of the user who logged in; this information is passed to the *getpwnam*( ) function to get the user database entry for that user.

```
#include <sys/types.h>
#include <pwd.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
...
char *lgn;
struct passwd *pw;
...
if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
    fprintf(stderr, "Get of user information failed.\n"); exit(1);
}
...
```


APPLICATION USAGE

Three names associated with the current process can be determined: *getpwuid*(*geteuid*( )) returns the name associated with the effective user ID of the process; *getlogin*( ) returns the name associated with the current login activity; and *getpwuid*(*getuid*( )) returns the name associated with the real user ID of the process.

The *getpwnam_r*( ) function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

Portable applications should take into account that it is usual for an implementation to return –1 from *sysconf*( ) indicating that there is no maximum for _SC_GETPW_R_SIZE_MAX.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getpwuid( ), *sysconf*( )

XBD **<pwd.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 1. Derived from System V Release 2.0.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

The *getpwnam_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *getpwnam*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *getpwnam_r*( ) function is marked as part of the Thread-Safe Functions option.

The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION describing matching the *name*.

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the buffer from *bufsize* characters to bytes.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-166 is applied.

The *getpwnam_r*( ) function is moved from the Thread-Safe Functions option to the Base.

A minor addition is made to the EXAMPLES section, reminding the application developer to free memory allocated as if by *malloc*( ).

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0255 [75,428] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0170 [808] and XSH/TC2-2008/0171 [656] are applied.

Issue 8

Austin Group Defect 398 is applied, changing the [ERANGE] error from ``may fail'' to ``shall fail''.

Austin Group Defect 1570 is applied, removing extra spacing in `"=="`.

39959 **NAME**

39960 getpwuid, getpwuid_r — search user database for a user ID

39961 **SYNOPSIS**

39962
```
#include <pwd.h>
```

39963
```
struct passwd *getpwuid(uid_t uid);
```
39964
```
int getpwuid_r(uid_t uid, struct passwd *pwd, char *buffer,
```
39965
```
    size_t bufsize, struct passwd **result);
```

39966 **DESCRIPTION**

39967 The *getpwuid*( ) function shall search the user database for an entry with a matching *uid*.

39968 The *getpwuid*( ) function need not be thread-safe.

39969 Applications wishing to check for error situations should set *errno* to 0 before calling *getpwuid*( ).
39970 If *getpwuid*( ) returns a null pointer and *errno* is set to non-zero, an error occurred.

39971 The *getpwuid_r*( ) function shall update the **passwd** structure pointed to by *pwd* and store a
39972 pointer to that structure at the location pointed to by *result*. The structure shall contain an entry
39973 from the user database with a matching *uid*. Storage referenced by the structure is allocated
39974 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to
39975 *sysconf* (_SC_GETPW_R_SIZE_MAX) returns either –1 without changing *errno* or an initial value
39976 suggested for the size of this buffer. A null pointer shall be returned at the location pointed to
39977 by *result* on error or if the requested entry is not found.

39978 **RETURN VALUE**

39979 The *getpwuid*( ) function shall return a pointer to a **struct passwd** with the structure as defined in
39980 **<pwd.h>** with a matching entry if found. A null pointer shall be returned if the requested entry
39981 is not found, or an error occurs. If the requested entry was not found, *errno* shall not be changed.
39982 On error, *errno* shall be set to indicate the error.

39983 The application shall not modify the structure to which the return value points, nor any storage
39984 areas pointed to by pointers within the structure. The returned pointer, and pointers within the
39985 structure, might be invalidated or the structure or the storage areas might be overwritten by a
39986 subsequent call to *getpwent*( ), *getpwnam*( ), or *getpwuid*( ). The returned pointer, and pointers
39987 within the structure, might also be invalidated if the calling thread is terminated.

39988 If successful, the *getpwuid_r*( ) function shall return zero; otherwise, an error number shall be
39989 returned to indicate the error.

39990 **ERRORS**

39991 These functions may fail if:

39992 [EIO] An I/O error has occurred.

39993 [EINTR] A signal was caught during *getpwuid*( ).

39994 [EMFILE] All file descriptors available to the process are currently open.

39995 [ENFILE] The maximum allowable number of files is currently open in the system.

39996 The *getpwuid_r*( ) function shall fail if:

39997 [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be
39998 referenced by the resulting **passwd** structure.

EXAMPLES

Note that *sysconf* (_SC_GETPW_R_SIZE_MAX) may return –1 if there is no hard limit on the size of the buffer needed to store all the groups returned. This example shows how an application can allocate a buffer of sufficient size to work with *getpwuid_r*( ).

```
long int initlen = sysconf(_SC_GETPW_R_SIZE_MAX);
size_t len;
if (initlen == -1)
    /* Default initial length. */
    len = 1024;
else
    len = (size_t) initlen;
struct passwd result;
struct passwd *resultp;
char *buffer = malloc(len);
if (buffer == NULL)
    ...handle error...
int e;
while ((e = getpwuid_r(42, &result, buffer, len, &resultp)) == ERANGE)
    {
    size_t newlen = 2 * len;
    if (newlen < len)
        ...handle error...
    len = newlen;
    char *newbuffer = realloc(buffer, len);
    if (newbuffer == NULL)
        ...handle error...
    buffer = newbuffer;
    }
if (e != 0)
    ...handle error...
free (buffer);
```

Getting an Entry for the Root User

The following example gets the user database entry for the user with user ID 0 (root).

```
#include <sys/types.h>
#include <pwd.h>
...
uid_t id = 0;
struct passwd *pwd;

pwd = getpwuid(id);
```


40038 **Finding the Name for the Effective User ID**

40039 The following example defines *pws* as a pointer to a structure of type **passwd**, which is used to
40040 store the structure pointer returned by the call to the *getpwuid*( ) function. The *geteuid*( ) function
40041 shall return the effective user ID of the calling process; this is used as the search criteria for the
40042 *getpwuid*( ) function. The call to *getpwuid*( ) shall return a pointer to the structure containing that
40043 user ID value.

```
40044 #include <unistd.h>
40045 #include <sys/types.h>
40046 #include <pwd.h>
40047 ...
40048 struct passwd *pws;
40049 pws = getpwuid(geteuid());
```

40050 **Finding an Entry in the User Database**

40051 The following example uses *getpwuid*( ) to search the user database for a user ID that was
40052 previously stored in a **stat** structure, then prints out the user name if it is found. If the user is not
40053 found, the program prints the numeric value of the user ID for the entry.

```
40054 #include <sys/types.h>
40055 #include <pwd.h>
40056 #include <stdio.h>
40057 ...
40058 struct stat statbuf;
40059 struct passwd *pwd;
40060 ...
40061 if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
40062     printf(" %-8.8s", pwd->pw_name);
40063 else
40064     printf(" %-8d", statbuf.st_uid);
```

40065 **APPLICATION USAGE**

40066 Three names associated with the current process can be determined: *getpwuid*(*geteuid*( )) returns
40067 the name associated with the effective user ID of the process; *getlogin*( ) returns the name
40068 associated with the current login activity; and *getpwuid*(*getuid*( )) returns the name associated
40069 with the real user ID of the process.

40070 The *getpwuid_r*( ) function is thread-safe and returns values in a user-supplied buffer instead of
40071 possibly using a static data area that may be overwritten by each call.

40072 Portable applications should take into account that it is usual for an implementation to return –1
40073 from *sysconf*( ) indicating that there is no maximum for _SC_GETPW_R_SIZE_MAX.

40074 **RATIONALE**

40075 None.

40076 **FUTURE DIRECTIONS**

40077 None.

40078 **SEE ALSO**

40079 *getpwnam*( ), *geteuid*( ), *getuid*( ), *getlogin*( ), *sysconf*( )

40080 XBD **<pwd.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 1. Derived from System V Release 2.0.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

The *getpwuid_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *getpwuid*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *getpwuid_r*( ) function is marked as part of the Thread-Safe Functions option.

The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION describing matching the *uid*.

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the buffer from *bufsize* characters to bytes.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-166 is applied.

The *getpwuid_r*( ) function is moved from the Thread-Safe Functions option to the Base.

A minor addition is made to the EXAMPLES section, reminding the application developer to free memory allocated as if by *malloc*( ).

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0256 [75] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0172 [808] and XSH/TC2-2008/0173 [656] are applied.

Issue 8

Austin Group Defect 398 is applied, changing the [ERANGE] error from ``may fail'' to ``shall fail''.

Austin Group Defect 1570 is applied, removing extra spacing in `"=="`.

NAME

getresgid — get real group ID, effective group ID, and saved set-group-ID

SYNOPSIS

XSI

```
#include <unistd.h>

int getresgid(gid_t *restrict rgid, gid_t *restrict egid,
    gid_t *restrict sgid);
```

DESCRIPTION

The *getresgid*( ) function shall store the real group ID, effective group ID, and saved set-group-ID of the calling process in the locations pointed to by the arguments *rgid*, *egid*, and *sgid*, respectively.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getresuid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setresgid*( ), *setresuid*( ), *setreuid*( ), *setuid*( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 8.

NAME

getresuid — get real user ID, effective user ID, and saved set-user-ID

SYNOPSIS

XSI

```
#include <unistd.h>

int getresuid(uid_t *restrict ruid, uid_t *restrict euid,
    uid_t *restrict suid);
```

DESCRIPTION

The *getresuid*( ) function shall store the real user ID, effective user ID, and saved set-user-ID of the calling process in the locations pointed to by the arguments *ruid*, *euid*, and *suid*, respectively.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getresgid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setresgid*( ), *setresuid*( ), *setreuid*( ), *setuid*( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 8.

40177 **NAME**

40178 getrlimit, setrlimit — control maximum resource consumption

40179 **SYNOPSIS**

40180 ```
#include <sys/resource.h>
```

40181 ```
int getrlimit(int resource, struct rlimit *rlp);
```
40182 ```
int setrlimit(int resource, const struct rlimit *rlp);
```

40183 **DESCRIPTION**

40184 The *getrlimit*( ) function shall get, and the *setrlimit*( ) function shall set, limits on the consumption
40185 of a variety of resources.

40186 Each call to either *getrlimit*( ) or *setrlimit*( ) identifies a specific resource to be operated upon as
40187 well as a resource limit. A resource limit is represented by an **rlimit** structure. The *rlim_cur*
40188 member specifies the current or soft limit and the *rlim_max* member specifies the maximum or
40189 hard limit. Soft limits can be changed by a process to any value that is less than or equal to the
40190 hard limit. A process can (irreversibly) lower its hard limit to any value that is greater than or
40191 equal to the soft limit. Only a process with appropriate privileges can raise a hard limit. Both
40192 hard and soft limits can be changed in a single call to *setrlimit*( ) subject to the constraints
40193 described above.

40194 The value RLIM_INFINITY, defined in **<sys/resource.h>**, shall be considered to be larger than
40195 any other limit value. If a call to *getrlimit*( ) returns RLIM_INFINITY for a resource, it means the
40196 implementation shall not enforce limits on that resource. Specifying RLIM_INFINITY as any
40197 resource limit value on a successful call to *setrlimit*( ) shall inhibit enforcement of that resource
40198 limit.

40199 The following resources are defined:

40200 RLIMIT_CORE — This is the maximum size of a file containing a core image, in bytes, that
40201 can be created by a process. A limit of 0 shall prevent the creation of such
40202 a file. If this limit is exceeded, the writing of a file containing a core image
40203 shall terminate at this size. Note that the production of such a file may be
40204 one of the implementation-defined actions for abnormal termination.

40205 XSI RLIMIT_CPU — This is the maximum amount of CPU time, in seconds, used by a process.
40206 If this limit is exceeded, SIGXCPU shall be generated for the process. If
40207 the process is catching or ignoring SIGXCPU, or all threads belonging to
40208 that process are blocking SIGXCPU, the behavior is unspecified.

40209 RLIMIT_DATA — This is the maximum size of a data segment of the process, in bytes. If
40210 this limit is exceeded, the *malloc*( ) function shall fail with *errno* set to
40211 [ENOMEM].

40212 RLIMIT_FSIZE — This is the maximum size of a file, in bytes, that can be created by a
40213 process. If a write or truncate operation would cause this limit to be
40214 XSI exceeded, a SIGXFSZ shall be generated for the thread; if the thread is
40215 blocking, or the process is catching or ignoring SIGXFSZ, the operation
40216 shall fail with an [EFBIG] error.

40217 RLIMIT_NOFILE — This is a number one greater than the maximum value that the system
40218 shall assign to a newly-created descriptor. If this limit is exceeded,
40219 functions that allocate a file descriptor shall fail with *errno* set to
40220 [EMFILE]. This limit constrains the number of file descriptors that a
40221 process can allocate.

  
```

RLIMIT_STACK — This is the maximum size of the initial thread's stack, in bytes. The implementation does not automatically grow the stack beyond this limit. If this limit is exceeded, SIGSEGV shall be generated for the thread. If the thread is blocking SIGSEGV, or the process is ignoring or catching SIGSEGV and has not made arrangements to use an alternate stack, the disposition of SIGSEGV shall be set to SIG_DFL before it is generated.

RLIMIT_AS — This is the maximum size of total available memory of the process, in bytes. If this limit is exceeded, the *malloc*( ) and *mmap*( ) functions shall fail with *errno* set to [ENOMEM]. In addition, the automatic stack growth fails with the effects outlined above.

When using the *getrlimit*( ) function, if a resource limit can be represented correctly in an object of type **rlim_t**, then its representation is returned; otherwise, if the value of the resource limit is equal to that of the corresponding saved hard limit, the value returned shall be RLIM_SAVED_MAX; otherwise, the value returned shall be RLIM_SAVED_CUR.

When using the *setrlimit*( ) function, if the requested new limit is RLIM_INFINITY, the new limit shall be ``no limit''; otherwise, if the requested new limit is RLIM_SAVED_MAX, the new limit shall be the corresponding saved hard limit; otherwise, if the requested new limit is RLIM_SAVED_CUR, the new limit shall be the corresponding saved soft limit; otherwise, the new limit shall be the requested value. In addition, if the corresponding saved limit can be represented correctly in an object of type **rlim_t** then it shall be overwritten with the new limit.

The result of setting a limit to RLIM_SAVED_MAX or RLIM_SAVED_CUR is unspecified unless a previous call to *getrlimit*( ) returned that value as the soft or hard limit for the corresponding resource limit.

The determination of whether a limit can be correctly represented in an object of type **rlim_t** is implementation-defined. For example, some implementations permit a limit whose value is greater than RLIM_INFINITY and others do not.

The *exec* family of functions shall cause resource limits to be saved.

RETURN VALUE

Upon successful completion, *getrlimit*( ) and *setrlimit*( ) shall return 0. Otherwise, these functions shall return –1 and set *errno* to indicate the error.

ERRORS

The *getrlimit*( ) and *setrlimit*( ) functions shall fail if:

[EINVAL] — An invalid *resource* was specified; or in a *setrlimit*( ) call, the new *rlim_cur* exceeds the new *rlim_max*.

[EPERM] — The limit specified to *setrlimit*( ) would have raised the maximum limit value, and the calling process does not have appropriate privileges.

The *setrlimit*( ) function may fail if:

[EINVAL] — The limit specified cannot be lowered because current usage is already higher than the limit.

EXAMPLES

None.

APPLICATION USAGE

If a process attempts to set the hard limit or soft limit for RLIMIT_NOFILE to less than the value of {_POSIX_OPEN_MAX} from **<limits.h>**, unexpected behavior may occur.

If a process attempts to set the hard limit or soft limit for RLIMIT_NOFILE to less than the highest currently open file descriptor +1, unexpected behavior may occur.

RATIONALE

These functions were previously part of the XSI option and have been moved to the Base so that portable shells, and other utilities that need to relay the wait status of a child process to a parent, can terminate themselves with the same signal that terminated the child but without overwriting a core image created by the child (through setting RLIMIT_CORE to zero, which disables core image creation). The RLIMIT_CPU and RLIMIT_FSIZE limits remain in the XSI option because they relate to other XSI functionality (SIGXCPU and SIGXFSZ).

It should be noted that RLIMIT_STACK applies ``at least'' to the stack of the initial thread in the process, and not to the sum of all the stacks in the process, as that would be very limiting unless the value is so big as to provide no value at all with a single thread.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fork*( ), *malloc*( ), *open*( ), *sigaltstack*( ), *sysconf*( )

XBD **<sys/resource.h>**

XCU *ulimit*

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

An APPLICATION USAGE section is added.

Large File Summit extensions are added.

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/25 is applied, changing wording for RLIMIT_NOFILE in the DESCRIPTION related to functions that allocate a file descriptor failing with [EMFILE]. Text is added to the APPLICATION USAGE section noting the consequences of a process attempting to set the hard or soft limit for RLIMIT_NOFILE less than the highest currently open file descriptor +1.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/46 is applied, updating the definition of RLIMIT_STACK in the DESCRIPTION from ``the maximum size of a process stack'' to ``the maximum size of the initial thread's stack''. Text is added to the RATIONALE section.

Issue 8

Austin Group Defects 51 and 1669 are applied, moving the *getrlimit*( ) and *setrlimit*( ) functions, excluding the RLIMIT_CPU limit, from the XSI option to the Base.

Austin Group Defect 1141 is applied, changing the description of RLIMIT_CORE.

Austin Group Defect 1416 is applied, changing some uses of ``may'' to ``can'' and one to ``shall''.

40304 Austin Group Defect 1418 is applied, changing the SEE ALSO section.

NAME

getrusage — get information about resource utilization

SYNOPSIS

XSI

```
#include <sys/resource.h>

int getrusage(int who, struct rusage *r_usage);
```

DESCRIPTION

The *getrusage*( ) function shall provide measures of the resources used by the current process or its terminated and waited-for child processes. If the value of the *who* argument is RUSAGE_SELF, information shall be returned about resources used by the current process. If the value of the *who* argument is RUSAGE_CHILDREN, information shall be returned about resources used by the children of the current process that have terminated and been waited-for and their children that have terminated and been waited-for, recursively. If the child is never waited for (for example, if the parent has SA_NOCLDWAIT set or sets SIGCHLD to SIG_IGN), the resource information for the child process is discarded and not included in the resource information provided by *getrusage*( ).

The *r_usage* argument is a pointer to an object of type **struct rusage** in which the returned information is stored.

RETURN VALUE

Upon successful completion, *getrusage*( ) shall return 0; otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *getrusage*( ) function shall fail if:

[EINVAL] The value of the *who* argument is not valid.

EXAMPLES

Using getrusage( )

The following example returns information about the resources used by the current process.

```
#include <sys/resource.h>
...
int who = RUSAGE_SELF;
struct rusage usage;
int ret;

ret = getrusage(who, &usage);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exit( ), *sigaction*( ), *time*( ), *times*( ), *wait*( )

XBD **<sys/resource.h>**

40346 **CHANGE HISTORY**

40347 First released in Issue 4, Version 2.

40348 **Issue 5**

40349 Moved from X/OPEN UNIX extension to BASE.

40350 **Issue 8**

40351 Austin Group Defect 1336 is applied, clarifying the requirements when the *who* argument is
40352 RUSAGE_CHILDREN.

40353 **NAME**

40354 getservbyname, getservbyport, getservent — network services database functions

40355 **SYNOPSIS**

40356 ```
#include <netdb.h>
```

40357 ```
struct servent *getservbyname(const char *name, const char *proto);
```
40358 ```
struct servent *getservbyport(int port, const char *proto);
```
40359 ```
struct servent *getservent(void);
```

40360 **DESCRIPTION**

40361 Refer to *endservent( )*.

40362 **NAME**

40363 getsid — get the process group ID of a session leader

40364 **SYNOPSIS**

40365 `#include <unistd.h>`

40366 `pid_t getsid(pid_t pid);`

40367 **DESCRIPTION**

40368 The *getsid*( ) function shall obtain the process group ID of the process that is the session leader of
40369 the process specified by *pid*. If *pid* is (**pid_t**)0, it specifies the calling process.

40370 **RETURN VALUE**

40371 Upon successful completion, *getsid*( ) shall return the process group ID of the session leader of
40372 the specified process. Otherwise, it shall return –1 and set *errno* to indicate the error.

40373 **ERRORS**

40374 The *getsid*( ) function shall fail if:

40375 [EPERM] The process specified by *pid* is not in the same session as the calling process,
40376 and the implementation does not allow access to the process group ID of the
40377 session leader of that process from the calling process.

40378 [ESRCH] There is no process with a process ID equal to *pid*.

40379 **EXAMPLES**

40380 None.

40381 **APPLICATION USAGE**

40382 None.

40383 **RATIONALE**

40384 None.

40385 **FUTURE DIRECTIONS**

40386 None.

40387 **SEE ALSO**

40388 *exec*, *fork*( ), *getpid*( ), *getpgid*( ), *setpgid*( ), *setsid*( )

40389 XBD **<unistd.h>**

40390 **CHANGE HISTORY**

40391 First released in Issue 4, Version 2.

40392 **Issue 5**

40393 Moved from X/OPEN UNIX extension to BASE.

40394 **Issue 7**

40395 The *getsid*( ) function is moved from the XSI option to the Base.

40396 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0258 [421] is applied.

40397 **NAME**

40398 getsockname — get the socket name

40399 **SYNOPSIS**

40400 ```
#include <sys/socket.h>
```

40401 ```
int getsockname(int socket, struct sockaddr *restrict address,
```

40402 ```
    socklen_t *restrict address_len);
```

40403 **DESCRIPTION**

40404 The *getsockname*( ) function shall retrieve the locally-bound name of the specified socket, store
40405 this address in the **sockaddr** structure pointed to by the *address* argument, and store the length of
40406 this address in the object pointed to by the *address_len* argument.

40407 The *address_len* argument points to a **socklen_t** object which on input specifies the length of the
40408 supplied **sockaddr** structure, and on output specifies the length of the socket address. If the
40409 actual length of the address is greater than the length of the supplied **sockaddr** structure, the
40410 stored address shall be truncated.

40411 If the socket has not been bound to a local name, the value stored in the object pointed to by
40412 *address* is unspecified.

40413 **RETURN VALUE**

40414 Upon successful completion, 0 shall be returned, the *address* argument shall point to the address
40415 of the socket, and the *address_len* argument shall point to the length of the address. Otherwise, –1
40416 shall be returned and *errno* set to indicate the error.

40417 **ERRORS**

40418 The *getsockname*( ) function shall fail if:

40419 [EBADF] The *socket* argument is not a valid file descriptor.

40420 [ENOTSOCK] The *socket* argument does not refer to a socket.

40421 [EOPNOTSUPP] The operation is not supported for this socket's protocol.

40422 The *getsockname*( ) function may fail if:

40423 [EINVAL] The socket has been shut down.

40424 [ENOBUFS] Insufficient resources were available in the system to complete the function.

40425 **EXAMPLES**

40426 None.

40427 **APPLICATION USAGE**

40428 For AF_UNIX sockets, it is recommended that *address* points to a buffer of length greater than
40429 `sizeof(struct sockaddr_un)` which has been initialized with null bytes. That way, even if
40430 the implementation supports the use of all bytes of *sun_path* without a terminating null byte, the
40431 larger buffer guarantees that the *sun_path* member can then be passed to other interfaces that
40432 expect a null-terminated string. If no truncation occurred based on the input value of *address_len*,
40433 it is unspecified whether the returned *address_len* will be `sizeof(struct sockaddr_un)`, or
40434 merely a value at least as large as `offsetof(struct sockaddr_un, sun_path)` plus the
40435 number of non-null bytes stored in *sun_path*.

40436 **RATIONALE**

40437 None.

  
```

FUTURE DIRECTIONS

None.

SEE ALSO

accept( ), *bind*( ), *getpeername*( ), *socket*( )

XBD **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The **restrict** keyword is added to the *getsockname*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0259 [464] is applied.

Issue 8

Austin Group Defect 561 is applied, adding a paragraph about *sun_path* to APPLICATION USAGE.

Austin Group Defect 1565 is applied, changing the description of *address_len*.

40453 **NAME**

40454 getsockopt — get the socket options

40455 **SYNOPSIS**

40456 ```
#include <sys/socket.h>
```

40457 ```
int getsockopt(int socket, int level, int option_name,
```

40458 ```
    void *restrict option_value, socklen_t *restrict option_len);
```

40459 **DESCRIPTION**

40460 The *getsockopt*( ) function manipulates options associated with a socket.

40461 The *getsockopt*( ) function shall retrieve the value for the option specified by the *option_name*
40462 argument for the socket specified by the *socket* argument. If the size of the option value is greater
40463 than *option_len*, the value stored in the object pointed to by the *option_value* argument shall be
40464 silently truncated. Otherwise, the object pointed to by the *option_len* argument shall be modified
40465 to indicate the actual length of the value.

40466 The *level* argument specifies the protocol level at which the option resides. To retrieve options at
40467 the socket level, specify the *level* argument as SOL_SOCKET. To retrieve options at other levels,
40468 supply the appropriate level identifier for the protocol controlling the option. For example, to
40469 indicate that an option is interpreted by the TCP (Transmission Control Protocol), set *level* to
40470 IPPROTO_TCP as defined in the **<netinet/in.h>** header.

40471 The socket in use may require the process to have appropriate privileges to use the *getsockopt*( )
40472 function.

40473 The *option_name* argument specifies a single option to be retrieved. It can be one of the socket-
40474 level options defined in **<sys/socket.h>** and described in Section 2.10.16 (on page 554).

40475 **RETURN VALUE**

40476 Upon successful completion, *getsockopt*( ) shall return 0; otherwise, –1 shall be returned and *errno*
40477 set to indicate the error.

40478 **ERRORS**

40479 The *getsockopt*( ) function shall fail if:

40480 [EBADF] The *socket* argument is not a valid file descriptor.

40481 [EINVAL] The specified option is invalid at the specified socket level.

40482 [ENOPROTOOPT]
40483 The option is not supported by the protocol.

40484 [ENOTSOCK] The *socket* argument does not refer to a socket.

40485 The *getsockopt*( ) function may fail if:

40486 [EACCES] The calling process does not have appropriate privileges.

40487 [EINVAL] The socket has been shut down.

40488 [ENOBUFS] Insufficient resources are available in the system to complete the function.

  
```

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.10.16 (on page 554), *bind*( ), *close*( ), *endprotoent*( ), *setsockopt*( ), *socket*( )

XBD **<sys/socket.h>**, **<netinet/in.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The **restrict** keyword is added to the *getsockopt*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/47 is applied, updating the description of SO_LINGER in the DESCRIPTION so that it blocks the calling thread rather than the process.

Issue 7

Austin Group Interpretation 1003.1-2001 #158 is applied, removing text relating to socket options that is now in Section 2.10.16 (on page 554).

40509 **NAME**

40510 getsubopt — parse suboption arguments from a string

40511 **SYNOPSIS**

40512 CX
```
#include <stdlib.h>

int getsubopt(char **restrict optionp,
    char * const *restrict keylistp, char **restrict valuep);
```

40515 **DESCRIPTION**

40516 The *getsubopt*( ) function shall parse suboption arguments in a flag argument. Such options often
40517 result from the use of *getopt*( ).

40518 The *getsubopt*( ) argument *optionp* is a pointer to a pointer to the option argument string. The
40519 suboption arguments shall be separated by <comma> characters and each may consist of either
40520 a single token, or a token-value pair separated by an <equals-sign>.

40521 The *keylistp* argument shall be a pointer to a vector of strings. The end of the vector is identified
40522 by a null pointer. Each entry in the vector is one of the possible tokens that might be found in
40523 **optionp*. Since <comma> characters delimit suboption arguments in *optionp*, they should not
40524 appear in any of the strings pointed to by *keylistp*. Similarly, because an <equals-sign> separates
40525 a token from its value, the application should not include an <equals-sign> in any of the strings
40526 pointed to by *keylistp*. The *getsubopt*( ) function shall not modify the *keylistp* vector.

40527 The *valuep* argument is the address of a value string pointer.

40528 If a <comma> appears in *optionp*, it shall be interpreted as a suboption separator. After <comma>
40529 characters have been processed, if there are one or more <equals-sign> characters in a suboption
40530 string, the first <equals-sign> in any suboption string shall be interpreted as a separator between
40531 a token and a value. Subsequent <equals-sign> characters in a suboption string shall be
40532 interpreted as part of the value.

40533 If the string at **optionp* contains only one suboption argument (equivalently, no <comma>
40534 characters), *getsubopt*( ) shall update **optionp* to point to the null character at the end of the
40535 string. Otherwise, it shall isolate the suboption argument by replacing the <comma> separator
40536 with a null character, and shall update **optionp* to point to the start of the next suboption
40537 argument. If the suboption argument has an associated value (equivalently, contains an <equals-
40538 sign>), *getsubopt*( ) shall update **valuep* to point to the value's first character. Otherwise, it shall
40539 set **valuep* to a null pointer. The calling application may use this information to determine
40540 whether the presence or absence of a value for the suboption is an error.

40541 Additionally, when *getsubopt*( ) fails to match the suboption argument with a token in the *keylistp*
40542 array, the calling application should decide if this is an error, or if the unrecognized option
40543 should be processed in another way.

40544 **RETURN VALUE**

40545 The *getsubopt*( ) function shall return the index of the matched token string, or –1 if no token
40546 strings were matched.

40547 **ERRORS**

40548 No errors are defined.

EXAMPLES

Parsing Suboptions

The following example uses the *getsubopt*( ) function to parse a *value* argument in the *optarg* external variable returned by a call to *getopt*( ).

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int do_all;
const char *type;
int read_size;
int write_size;
int read_only;

enum
{
    RO_OPTION = 0,
    RW_OPTION,
    READ_SIZE_OPTION,
    WRITE_SIZE_OPTION
};

const char *mount_opts[] =
{
    [RO_OPTION] = "ro",
    [RW_OPTION] = "rw",
    [READ_SIZE_OPTION] = "rsize",
    [WRITE_SIZE_OPTION] = "wsize",
    NULL
};

int
main(int argc, char *argv[])
{
    char *subopts, *value;
    int opt;

    while ((opt = getopt(argc, argv, "at:o:")) != -1)
        switch(opt)
            {
            case 'a':
                do_all = 1;
                break;
            case 't':
                type = optarg;
                break;
            case 'o':
                subopts = optarg;
                while (*subopts != ' ')
                {
                    char *saved = subopts;
                    switch(getsubopt(&subopts, (char **)mount_opts,
```



```
                                &value))
                            {
                            case RO_OPTION:
                                read_only = 1;
                                break;
                            case RW_OPTION:
                                read_only = 0;
                                break;
                            case READ_SIZE_OPTION:
                                if (value == NULL)
                                    abort();
                                read_size = atoi(value);
                                break;
                            case WRITE_SIZE_OPTION:
                                if (value == NULL)
                                    abort();
                                write_size = atoi(value);
                                break;
                            default:
                                /* Unknown suboption. */
                                printf("Unknown suboption `%s'\n", saved);
                                abort();
                            }
                        }
                        break;
                    default:
                        abort();
                    }
            /* Do the real work. */

            return 0;
        }
```

If the above example is invoked with:

```
program -o ro,rsize=512
```

then after option parsing, the variable *do_all* will be 0, *type* will be a null pointer, *read_size* will be 512, *write_size* will be 0, and *read_only* will be 1. If it is invoked with:

```
program -o oops
```

it will print:

```
"Unknown suboption `oops'"
```

before aborting.

APPLICATION USAGE

The value of **valuep* when *getsubopt*( ) returns –1 is unspecified. Historical implementations provide various incompatible extensions to allow an application to access the suboption text that was not found in the *keylistp* array.

40639 **RATIONALE**

40640 The *keylistp* argument of *getsubopt*( ) is typed as **char** * **const** * to match historical practice.
40641 However, the standard is clear that implementations will not modify either the array or the
40642 strings contained in the array, as if the argument had been typed **const char** * **const** *.

40643 **FUTURE DIRECTIONS**

40644 None.

40645 **SEE ALSO**

40646 *getopt*( )

40647 XBD **<stdlib.h>**

40648 **CHANGE HISTORY**

40649 First released in Issue 4, Version 2.

40650 **Issue 5**

40651 Moved from X/OPEN UNIX extension to BASE.

40652 **Issue 6**

40653 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/26 is applied, correcting an editorial error
40654 in the SYNOPSIS.

40655 **Issue 7**

40656 The *getsubopt*( ) function is moved from the XSI option to the Base.

40657 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0260 [196], XSH/TC1-2008/0261 [196],
40658 XSH/TC1-2008/0262 [196], XSH/TC1-2008/0263 [196], XSH/TC1-2008/0264 [196],
40659 XSH/TC1-2008/0265 [196], XSH/TC1-2008/0266 [196], XSH/TC1-2008/0267 [196],
40660 XSH/TC1-2008/0268 [196], and XSH/TC1-2008/0269 [196] are applied.

40661 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0174 [791] is applied.

40662 **Issue 8**

40663 Austin Group Defect 444 is applied, adding the **restrict** keyword to the *getsubopt*( ) prototype.

NAME

dgettext, dgettext_l, dcgettext, dcgettext_l, gettext, gettext_l, ngettext, ngettext_l, dngettext, dngettext_l, dcngettext, dcngettext_l — message handling functions

SYNOPSIS

```
#include <libintl.h>

char *dgettext(const char *domainname, const char *msgid);
char *dgettext_l(const char *domainname, const char *msgid,
    locale_t locale);
char *dcgettext(const char *domainname, const char *msgid,
    int category);
char *dcgettext_l(const char *domainname, const char *msgid,
    int category, locale_t locale);
char *dngettext(const char *domainname, const char *msgid,
    const char *msgid_plural, unsigned long int n);
char *dngettext_l(const char *domainname, const char *msgid,
    const char *msgid_plural, unsigned long int n,
    locale_t locale);
char *dcngettext(const char *domainname, const char *msgid,
    const char *msgid_plural, unsigned long int n,
    int category);
char *dcngettext_l(const char *domainname, const char *msgid,
    const char *msgid_plural, unsigned long int n,
    int category, locale_t locale);
char *gettext(const char *msgid);
char *gettext_l(const char *msgid, locale_t locale);
char *ngettext(const char *msgid, const char *msgid_plural,
    unsigned long int n);
char *ngettext_l(const char *msgid, const char *msgid_plural,
    unsigned long int n, locale_t locale);
```

DESCRIPTION

The *gettext*( ) function shall:

- attempt to locate a suitable messages object (described in detail below) for the *LC_MESSAGES* category in the current locale, and for the current text domain (see *bindtextdomain*( )), containing the string identified by *msgid*,
- retrieve the string identified by *msgid* from the messages object,
- convert the string to the output codeset if necessary (described in detail below), and
- return the result.

If the locale name in effect is `"POSIX"` or `"C"` (i.e. the name associated with the *LC_MESSAGES* locale category in the current locale), or if no suitable messages object exists, or if no string identified by *msgid* exists in the messages object, or if an error occurs, *msgid* shall be returned.

The *dgettext*( ) function shall be equivalent to *gettext*( ), except *domainname* shall be used instead of the current text domain to locate the messages object.

The *dcgettext*( ) function shall be equivalent to *dgettext*( ), except the locale category identified by *category* shall be used instead of *LC_MESSAGES*.

The *ngettext*( ) function shall be equivalent to *gettext*( ), except:

- The string to retrieve shall be identified by a combination of *msgid* and *n* (see *msgfmt*).
- If the locale name in effect is "POSIX" or "C", or if no suitable messages object exists, or if no string identified by the combination of *msgid* and *n* exists in the messages object, or if an error occurs, the return value shall be *msgid* if *n* is 1, otherwise *msgid_plural*.

The *dngettext*() function shall be equivalent to *ngettext*(), except *domainname* shall be used instead of the current text domain to locate the messages object.

The *dcngettext*() function shall be equivalent to *dngettext*(), except the locale category identified by *category* shall be used instead of *LC_MESSAGES*.

The **_l*() functions shall be equivalent to their counterparts without the *_l* suffix, except *locale* shall be used instead of the current locale. If *locale* is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle, the behavior is undefined.

The application shall ensure that the *msgid* and *msgid_plural* arguments are strings. If either *msgid* or *msgid_plural* is an empty string, or contains characters not in the portable character set, the results are unspecified. If the *category* argument is *LC_ALL*, the results are unspecified.

The location of the messages object shall be determined according to the following criteria, stopping when the first messages object is found:

XSI
1. If the *NLSPATH* environment variable is set to a non-empty string, an *NLSPATH* search shall be performed as described in XBD Section 8.2 (on page 169). If *NLSPATH* identifies more than one template to use, each template in turn shall be used until a valid messages object is found.
2. If the *LANGUAGE* environment variable is set to a non-empty string, a *LANGUAGE* search shall be performed as described below. If *LANGUAGE* identifies more than one directory to search, each directory shall be searched until a valid messages object is found.
3. A single-locale search shall be performed as described below.

XSI For the NLSPATH search and the single-locale search, the single locale name used to locate the messages object shall be the locale name associated with the selected locale category from the current locale, or the provided locale object if calling one of the **_l*() functions; additional searches of locale names without *.codeset* (if present), without *_territory* (if present), and without *@modifier* (if present) may be performed.

For the *LANGUAGE* search, the value of the *LANGUAGE* environment variable shall be a list of one or more locale names separated by a <colon> (':') character. Each locale name shall be tried in the specified order. If a messages object for the locale does not exist, or cannot be opened, or is unsuitable for implementation-defined reasons (such as security), the next locale name (if any) shall be tried. If:

- a messages object for the locale can be opened but cannot be processed without error, or
- the messages object does not contain a string identified by *msgid*, or *msgid* and *n* for the *ngettext* functions,

it is unspecified whether the next locale name (if any) is tried. In all other cases, the messages object for the locale shall be used.

For each locale name in *LANGUAGE*, or if *LANGUAGE* is not set or is empty, or no suitable messages object is found in processing *LANGUAGE*, the pathname used to locate the messages object shall be *dirname*/*localename*/*categoryname*/*textdomainname***.mo**, where:

- The *dirname* part is the *dirname* argument of the most recent successful call to *bindtextdomain*() that had *textdomainname* as the *domainname* argument; any trailing <slash> characters in *dirname* shall be discarded. If a successful call to *bindtextdomain*() has not been made for *textdomainname*, an implementation-defined default directory shall be used.
- For the *LANGUAGE* search, the *localename* part is each locale name from *LANGUAGE* in turn; if a locale name has the format *language*[*_territory*][*.codeset*][*@modifier*], additional searches of locale names without *.codeset* (if present), without *_territory* (if present), and without *@modifier* (if present) may be performed; if *.codeset* is not present, additional searches of locale names with an added *.codeset* may be performed. For the single-locale search, the *localename* part is the name of the current locale, or the locale specified in an **_l*() function call, for the category named by *categoryname*. Spellings of codeset names are not standardized, and implementations may attempt to use different commonly known spellings, for example `"utf8"` and `"UTF-8"`.
- The *categoryname* part is the string `"LC_MESSAGES"` if *gettext*(), *dgettext*(), *ngettext*(), or *dngettext*() is called, or the locale category name corresponding to the *category* argument to *dcgettext*() or *dcngettext*(). Likewise for the **_l*() variants of all these functions.
- For *gettext*(), *gettext_l*(), *ngettext*(), and *ngettext_l*(), the *textdomainname* part is the text domain set by the last successful call to *textdomain*(). For *dgettext*(), *dcgettext*(), *dngettext*(), *dcngettext*(), and the **_l*() variants of these functions, *textdomainname* is the text domain specified by the *domainname* argument. The *domainname* argument shall be equivalent in syntax and meaning to the *domainname* argument to *textdomain*(), except that the selection of the text domain shall affect only the *dgettext*(), *dcgettext*(), *dngettext*(), and *dcngettext*() function calls and their **_l*() variants. If the *domainname* argument is a null pointer, the text domain set by the last successful call to *textdomain*() shall be used. For all of these functions, if a successful call to *textdomain*() has not been made the default text domain `"messages"` shall be used.

Resolution of the messages object pathname shall be performed the first time one of the *gettext* family of functions is called for a given combination of *dirname*, *localename*, *categoryname*, and *textdomainname*. It is unspecified whether the pathname is re-resolved if the combination has been used before in a call to one of the *gettext* family of functions. If *bindtextdomain*() performs pathname resolution of its *dirname* argument, only the part of the messages object pathname after *dirname* shall be resolved by the *gettext* family of functions.

When one of the *gettext* family of functions returns a message string that was found in a messages object, it shall convert the codeset of the message string to the output codeset if a codeset is specified in the messages object (see *msgfmt*) and the output codeset is not the same as that codeset. If a successful call to *bind_textdomain_codeset*() has been made with the text domain of the messages object as the *domainname* argument and a non-null *codeset* argument, the output codeset shall be the *codeset* argument from the most recent such call. Otherwise, the output codeset shall be the codeset of characters in the current locale, or the provided locale object if calling one of the **_l*() functions, as specified by the *LC_CTYPE* category of the locale. The conversion shall be performed as if by a call to *iconv*() using a conversion descriptor returned by *iconv_open*(<*output codeset*>, <*messages object codeset*>), except that if the return value of *iconv*() would be greater than zero, the non-identical conversions performed by the *gettext* family of functions need not be the same as those that such an *iconv*() call would perform. If an error prevents the codeset conversion from being performed, the *gettext* family of functions shall behave as if no message string was found in the messages object. If at least one non-identical conversion is performed that results in a fallback character (one that does not provide any information about the character it was converted from, for example, a <question-mark> or ``replacement-character''), the *gettext* family of functions may behave as if no message string was

found in the messages object.

RETURN VALUE

The *gettext*(), *gettext_l*(), *dgettext*(), *dgettext_l*(), *dcgettext*(), and *dcgettext_l*() functions shall return the message string described in DESCRIPTION if successful. Otherwise, they shall return *msgid*.

The *ngettext*(), *ngettext_l*(), *dngettext*(), *dngettext_l*(), *dcngettext*(), and *dcngettext_l*() functions shall return the message string described in DESCRIPTION if successful. Otherwise, *msgid* shall be returned if *n* is equal to 1, or *msgid_plural* if *n* is not equal to 1.

The application shall ensure that it does not modify the returned string. A subsequent call to a *gettext* family function shall not overwrite or invalidate the returned string. The returned string may be invalidated by a subsequent call to *bind_textdomain_codeset*(), *bindtextdomain*(), *setlocale*(), or *textdomain*() in the same process, except for calls that only query values. The returned string shall not be invalidated by a subsequent call to *uselocale*().

ERRORS

The *gettext* family of functions shall not modify *errno*. If an error occurs these functions shall return a string as described in RETURN VALUE.

EXAMPLES

The example code below assumes the following:

- The implementation-defined default directory is **/system/gettextlib**.
- The following locales are available on the target system: en_US, en_GB, de_DE. The codeset used for all of these locales is UTF-8.
- The en_AU locale is not available on the target system.
- The target system supports conversion from ISO/IEC 8859-1 to UTF-8.
- The codeset used for the POSIX locale is ASCII.
- The target system does not support conversion from ISO/IEC 8859-1 to ASCII.

Furthermore, the following **.mo** files (and only the following **.mo** files) are installed:

- **/system/gettextlib/en_US/LC_MESSAGES/mail.mo**
- **/messagecatalogs/example/en_US/LC_MESSAGES/mail.mo**

These are compiled from a portable messages object source file (dot-po file) with the following ISO/IEC 8859-1 encoded contents (see the EXTENDED DESCRIPTION of the *msgfmt* utility for a description of the dot-po file format):

```
msgid ""
msgstr ""
"Content-Type: text/plain; charset=ISO_8859-1\n"
"Plural-Forms: nplurals=4; plural= n==1?0: (n>1&&n<10)?1: (n==0)?2:3;\n"
msgid "recipient"
msgid_plural "recipients"
msgstr[0] "1 recipient"
msgstr[1] "2 to 9 recipients"
msgstr[2] "no recipients"
msgstr[3] "more than 9 recipients"
```

/system/gettextlib/de_DE/LC_MESSAGES/mail.mo is compiled from a dot-po file with the following ISO/IEC 8859-1 encoded contents:


```
msgid ""
msgstr ""
"Content-Type: text/plain; charset=ISO_8859-1\n"
"Plural-Forms: nplurals=4; plural= n==1?0: (n>1&&n<5)?1: (n==0)?2:3;\n"
msgid "recipient"
msgid_plural "recipients"
msgstr[0] "1 Empfänger"
msgstr[1] "2 bis 4 Empfänger"
msgstr[2] "keine Empfänger"
msgstr[3] "mehr als 4 Empfänger"
```

/messagecatalogs/example/en_GB/LC_MESSAGES/mail.mo is compiled from a dot-po file with the following ISO/IEC 8859-1 encoded contents:

```
msgid ""
msgstr ""
"Content-Type: text/plain; charset=ISO_8859-1\n"
"Plural-Forms: nplurals=4; plural= n==1?0: (n>1&&n<5)?1: (n==0)?2:3;\n"
msgid "recipient"
msgid_plural "recipients"
msgstr[0] "1 recipient"
msgstr[1] "2 to 4 recipients"
msgstr[2] "no recipients"
msgstr[3] "5 or more recipients"
```

/messagecatalogs/example2/en_US/LC_MESSAGES/othermail.mo is not a suitable messages object file or is a suitable messages object file that does not contain the **msgid** `"recipient"`.

The following example demonstrates the interactions between *bindtextdomain*(), *bind_textdomain_codeset*(), *textdomain*(), and the *gettext* family of functions.

```
unsigned long n_recipients;
// strdup() is used to prevent default_domain from being invalidated by
// a future call to bindtextdomain()
const char *default_domain = strdup(bindtextdomain("mail", NULL));
setlocale(LC_MESSAGES, "POSIX");
setlocale(LC_CTYPE, "POSIX");

n_recipients = 1;
// The following outputs "recipient" with the same encoding as the
// "recipient" argument to ngettext():
printf("%s\n", ngettext("recipient", "recipients", n_recipients));

n_recipients = 3;
// The following outputs "recipients" with the same encoding as the
// "recipients" argument to ngettext():
printf("%s\n", ngettext("recipient", "recipients", n_recipients));

setlocale(LC_MESSAGES, "en_US");
setlocale(LC_CTYPE, "en_US");
textdomain("mail");

n_recipients = 1;
// The following outputs "1 recipient", encoded in UTF-8:
printf("%s\n", ngettext("recipient", "recipients", n_recipients));

n_recipients = 3;
```



```
40890         // The following outputs "2 to 9 recipients", encoded in UTF-8:
40891         printf("%s\n", ngettext("recipient", "recipients", n_recipients));
40892         setlocale(LC_MESSAGES, "en_GB");
40893         setlocale(LC_CTYPE, "en_GB");
40894         bindtextdomain("mail", "/messagecatalogs/example/");
40895         n_recipients = 3;
40896         // The following outputs "2 to 4 recipients", encoded in UTF-8:
40897         printf("%s\n", ngettext("recipient", "recipients", n_recipients));
40898         setlocale(LC_MESSAGES, "en_US");
40899         setlocale(LC_CTYPE, "en_US");
40900         textdomain("othermail");
40901         bindtextdomain("othermail", "/messagecatalogs/example2/");
40902         n_recipients = 3;
40903         // The following outputs "recipients" with the same encoding as the
40904         // "recipients" argument to ngettext():
40905         printf("%s\n", ngettext("recipient", "recipients", n_recipients));
40906         // Because there is no locale named en_AU on the system, en_US is used:
40907         setenv("LANGUAGE", "en_AU:en_US:en_GB", 1);
40908         setlocale(LC_MESSAGES, "");
40909         setlocale(LC_CTYPE, "");
40910         bindtextdomain("mail", default_domain);
40911         // The following outputs "2 to 9 recipients", encoded in UTF-8:
40912         printf("%s\n", dngettext("mail", "recipient", "recipients", 3));
40913         textdomain("mail");
40914         bind_textdomain_codeset("mail", "UTF-8");
40915         setlocale(LC_MESSAGES, "de_DE");
40916         setlocale(LC_CTYPE, "de_DE");
40917         // Clear the LANGUAGE environment variable, otherwise it would take
40918         // precedence over the locale set above, and en_US would continue to
40919         // be used.
40920         setenv("LANGUAGE", "", 1);
40921         n_recipients = 1;
40922         // The following outputs "1 Empfänger", encoded in UTF-8:
40923         printf("%s\n", ngettext("recipient", "recipients", n_recipients));
40924         bind_textdomain_codeset("mail", "ASCII");
40925         setlocale(LC_CTYPE, "POSIX");
40926         n_recipients = 1;
40927         // The following outputs "recipient" with the same encoding as the
40928         // "recipient" argument to ngettext() - remember, the system is assumed
40929         // to not support conversion from ISO/IEC 8859-1 to ASCII:
40930         printf("%s\n", ngettext("recipient", "recipients", n_recipients));
40931         free(default_domain);
```


APPLICATION USAGE

These functions do not impose a limit on message length. Note that translated strings typically have a different length than the input strings, possibly much longer, and applications using these translations in formatted text (for example, aligned columns for a table) should take that into account.

The *dcgettext*(), *dcgettext_l*(), *dcngettext*(), and *dcngettext_l*() functions are useful to retrieve locale-specific strings for a category other than *LC_MESSAGES*. For example, they can be used to obtain a time format string from the *LC_TIME* category; because the locale setting of *LC_TIME* and *LC_MESSAGES* can be different, using the other *gettext* family functions in such a case might cause an undesired result. All of the functions in the *gettext* family of functions, except *dcgettext*(), *dcgettext_l*(), *dcngettext*(), and *dcngettext_l*(), search for messages objects only in the *LC_MESSAGES* category.

Implementations typically, but are not required to, *mmap*() the messages object file the first time one of the *gettext* family of functions is called, and keep that map in place until it is no longer expected to be used. For example, a successful call to *bindtextdomain*() will typically cause the next call to one of the *gettext* family of functions to *munmap*() the previous file and *mmap*() the new file. Applications should not rely on this behavior, however: the implementation is allowed to cache previously used maps, or not use *mmap*() at all and reopen the file each time one of the *gettext* family of functions is called.

The *msgid* and *msgid_plural* arguments are typically in (US) English. The arguments are always used in the POSIX or C locale, and when a *gettext* family function encounters an error, so they should not be abstract message identifiers (for example, `"message 123"`) and they should only use characters in the portable character set (to avoid outputting byte sequences that are not valid characters in the current output codeset). If the *xgettext* utility is used to extract the *msgid* and *msgid_plural* arguments from C source files into a template dot-po file, the arguments must be string literals in order for the resulting file to be useful to translators.

The strings returned by the *gettext* family of functions are not guaranteed to contain only characters that are valid in the current output codeset. In particular, byte sequences that do not form valid characters can occur when:

- The *msgid* or *msgid_plural* arguments use characters outside the portable character set.
- The messages object file does not specify a character set and uses characters outside the portable character set.

The strings returned by the *gettext* family of functions are guaranteed to remain valid until invalidated as described in the RETURN VALUE section. This includes strings that are created by codeset conversion; those strings are freed by the implementation, not the application. Thus, it is safe to call *gettext* family functions multiple times in situations such as:

```
printf("%s %s\n", gettext("foo"), gettext("bar"));
```

RATIONALE

Although the return type of these functions ought to be **const char** *, it is **char** * to match historical practice.

The *gettext* family of functions is frequently used in reporting errors. In fact, it is possible to have an application that attempts to create an error message that combines a translated string via *gettext*() with an error string provided by *strerror*(). The standard requires that the *gettext* family of functions does not modify *errno*, so that an application need not worry about complications of providing sequencing points to capture a stable value of *errno* prior to the translation of the error message, and so that the user will still get a somewhat useful string (even if it is the untranslated original string) on any failure.

There are no wide character equivalents for these functions; historically no implementation is known to exist, and the multi-byte message returned from these functions can, in most instances, be converted to wide characters by the application if desired.

Some historical *gettext* implementations returned the translated string from the messages object without codeset conversion if *iconv_open*( ) fails. This is considered to be a bug in those implementations.

FUTURE DIRECTIONS

None.

SEE ALSO

bindtextdomain( ), *catopen*( ), *iconv*( ), *setlocale*( ), *uselocale*( )

XBD **<libintl.h>**, **<limits.h>**

XCU *gettext*, *msgfmt*, *xgettext*

CHANGE HISTORY

First released in Issue 8.

NAME

getuid — get a real user ID

SYNOPSIS

```
#include <unistd.h>

uid_t getuid(void);
```

DESCRIPTION

The *getuid*( ) function shall return the real user ID of the calling process. The *getuid*( ) function shall not modify *errno*.

RETURN VALUE

The *getuid*( ) function shall always be successful and no return value is reserved to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

Setting the Effective User ID to the Real User ID

The following example sets the effective user ID of the calling process to the real user ID.

```
#include <unistd.h>
...
seteuid(getuid());
```

APPLICATION USAGE

None.

RATIONALE

In a conforming environment, *getuid*( ) will always succeed. It is possible for implementations to provide an extension where a process in a non-conforming environment will not be associated with a user or group ID. It is recommended that such implementations return (**uid_t**)–1 and set *errno* to indicate such an environment; doing so does not violate this standard, since such an environment is already an extension.

FUTURE DIRECTIONS

None.

SEE ALSO

getegid( ), *geteuid*( ), *getgid*( ), *getresgid*( ), *getresuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setresgid*( ), *setresuid*( ), *setreuid*( ), *setuid*( )

XBD **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

41035 **Issue 7**

41036 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0175 [511] and XSH/TC2-2008/0176
41037 [897] are applied.

41038 **Issue 8**

41039 Austin Group Defect 1344 is applied, adding *getresgid*( ), *getresuid*( ), *setresgid*( ), and *setresuid*( ) to
41040 SEE ALSO.

41041 **NAME**

41042 getutxent, getutxid, getutxline — get user accounting database entries

41043 **SYNOPSIS**

41044 XSI
```
#include <utmpx.h>

struct utmpx *getutxent(void);
struct utmpx *getutxid(const struct utmpx *id);
struct utmpx *getutxline(const struct utmpx *line);
```

41048 **DESCRIPTION**

41049 Refer to *endutxent( )*.

NAME

getwc — get a wide character from a stream

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

wint_t getwc(FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *getwc*( ) function shall be equivalent to *fgetwc*( ), except that if it is implemented as a macro it may evaluate *stream* more than once, so the argument should never be an expression with side-effects.

RETURN VALUE

Refer to *fgetwc*( ).

ERRORS

Refer to *fgetwc*( ).

EXAMPLES

None.

APPLICATION USAGE

Since it may be implemented as a macro, *getwc*( ) may treat incorrectly a *stream* argument with side-effects. In particular, *getwc*(*f++) does not necessarily work as expected. Therefore, use of this function is not recommended; *fgetwc*( ) should be used instead.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fgetwc*( )

XBD **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working draft.

Issue 5

The Optional Header (OH) marking is removed from **<stdio.h>**.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0270 [14] is applied.

NAME

getwchar — get a wide character from a *stdin* stream

SYNOPSIS

```
#include <wchar.h>

wint_t getwchar(void);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *getwchar*( ) function shall be equivalent to *getwc*(*stdin*).

RETURN VALUE

Refer to *fgetwc*( ).

ERRORS

Refer to *fgetwc*( ).

EXAMPLES

None.

APPLICATION USAGE

If the **wint_t** value returned by *getwchar*( ) is stored into a variable of type **wchar_t** and then compared against the **wint_t** macro WEOF, the result may be incorrect. Only the **wint_t** type is guaranteed to be able to represent any wide character and WEOF.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fgetwc*( ), *getwc*( )

XBD **<wchar.h>**

CHANGE HISTORY

First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working draft.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0271 [14] is applied.

NAME

glob, globfree — generate pathnames matching a pattern

SYNOPSIS

```
#include <glob.h>

int glob(const char *restrict pattern, int flags,
    int(*errfunc)(const char *epath, int eerrno),
    glob_t *restrict pglob);
void globfree(glob_t *pglob);
```

DESCRIPTION

The *glob*( ) function is a pathname generator that shall implement the rules defined in XCU Section 2.14 (on page 2523), with optional support for rule 3 in XCU Section 2.14.3 (on page 2525).

The structure type **glob_t** is defined in **<glob.h>** and includes at least the following members:

Member Type	Member Name	Description
size_t	*gl_pathc*	Count of paths matched by *pattern*.
**char ** **	*gl_pathv*	Pointer to a list of matched pathnames.
size_t	*gl_offs*	Slots to reserve at the beginning of *gl_pathv*.

The argument *pattern* is a pointer to a pathname pattern to be expanded. The *glob*( ) function shall match all accessible pathnames against this pattern and develop a list of all pathnames that match. In order to have access to a pathname, *glob*( ) requires search permission on every component of a path except the last, and read permission on each directory of any filename component of *pattern* that contains any of the following special characters: `'*'`, `'?'`, and `'['`.

The *glob*( ) function shall store the number of matched pathnames into *pglob*->*gl_pathc* and a pointer to a list of pointers to pathnames into *pglob*->*gl_pathv*. The pathnames shall be in sort order as defined by the current setting of the *LC_COLLATE* category; see XBD Section 7.3.2 (on page 139). The first pointer after the last pathname shall be a null pointer. If the pattern does not match any pathnames, the returned number of matched paths is set to 0, and the contents of *pglob*->*gl_pathv* are implementation-defined.

It is the caller's responsibility to create the structure pointed to by *pglob*. The *glob*( ) function shall allocate other space as needed, including the memory pointed to by *gl_pathv*. The *globfree*( ) function shall free any space associated with *pglob* from a previous call to *glob*( ). The *globfree*( ) function shall not modify *errno* if *pglob* was previously used by *glob*( ) and not yet freed.

The *flags* argument is used to control the behavior of *glob*( ). The value of *flags* is a bitwise-inclusive OR of zero or more of the following constants, which are defined in **<glob.h>**:

GLOB_APPEND — Append pathnames generated to the ones from a previous call to *glob*( ).

GLOB_DOOFFS — Make use of *pglob*->*gl_offs*. If this flag is set, *pglob*->*gl_offs* is used to specify how many null pointers to add to the beginning of *pglob*->*gl_pathv*. In other words, *pglob*->*gl_pathv* shall point to *pglob*->*gl_offs* null pointers, followed by *pglob*->*gl_pathc* pathname pointers, followed by a null pointer.

GLOB_ERR — Cause *glob*( ) to return when an attempt to open or search a pathname as a directory, or an attempt to read an opened directory, fails because of an error condition that is related to file system contents and prevents *glob*( ) from expanding the pattern. If this flag is not set, *glob*( ) shall not treat such conditions as an error, and shall continue to look for matches. Other

error conditions may also be treated the same way as error conditions that are related to file system contents.

GLOB_MARK — For each pathname that matches *pattern* and is determined to be a directory after pathname resolution, process the pathname so the result is as if the following steps are applied in order:

1. If the pathname is <slash>, do not modify the pathname and skip the remaining steps.
2. If the pathname is <slash><slash> and the implementation handles pathname resolution of a pathname starting with exactly two successive <slash> characters differently than it handles a pathname starting with only a single <slash>, do not modify the pathname and skip the remaining steps.
3. If the pathname does not end with a <slash>, append a <slash> to the pathname and skip the remaining steps.
4. A <slash> may be appended to the pathname.
5. If there are multiple <slash> characters at the end of the pathname, all but one of those trailing <slash> characters may be removed from the pathname.

GLOB_NOCHECK — Supports rule 3 in XCU Section 2.14.3 (on page 2525). If *pattern* does not match any pathname, then *glob*( ) shall return a list consisting of only *pattern*, and the number of matched pathnames is 1.

GLOB_NOESCAPE — Disable backslash escaping.

GLOB_NOSORT — Ordinarily, *glob*( ) sorts the matching pathnames according to the current setting of the *LC_COLLATE* category; see XBD Section 7.3.2 (on page 139). When this flag is used, the order of pathnames returned is unspecified.

The GLOB_APPEND flag can be used to append a new set of pathnames to those found in a previous call to *glob*( ). The following rules apply to applications when two or more calls to *glob*( ) are made with the same value of *pglob* and without intervening calls to *globfree*( ):

1. The first such call shall not set GLOB_APPEND. All subsequent calls shall set it.
2. All the calls shall set GLOB_DOOFFS, or all shall not set it.
3. After the second call, *pglob*−>*gl_pathv* points to a list containing the following:
   a. Zero or more null pointers, as specified by GLOB_DOOFFS and *pglob*−>*gl_offs*.
   b. Pointers to the pathnames that were in the *pglob*−>*gl_pathv* list before the call, in the same order as before.
   c. Pointers to the new pathnames generated by the second call, in the specified order.
4. The count returned in *pglob*−>*gl_pathc* shall be the total number of pathnames from the two calls.
5. The application can change any of the fields after a call to *glob*( ). If it does, the application shall reset them to the original value before a subsequent call, using the same *pglob* value, to *globfree*( ) or *glob*( ) with the GLOB_APPEND flag.

If *errfunc* is not a null pointer and, during the search, an attempt to open or search a pathname as a directory, or an attempt to read an opened directory, fails because of an error condition that

prevents *glob*( ) from expanding the pattern, *glob*( ) calls (**errfunc*( )) with two arguments:

1. The *epath* argument is a pointer to the path that failed.
2. The *eerrno* argument is the value of *errno* from the failure, as set by *opendir*( ), *readdir*( ), or *stat*( ). (Other values may be used to report other errors not explicitly documented for those functions.)

If (**errfunc*( )) is called and returns non-zero, or, optionally, if *errfunc* is a null pointer and the attempt failed because of an error condition that is not related to file system contents, or if the GLOB_ERR flag is set in *flags*, *glob*( ) shall stop the scan and return GLOB_ABORTED after setting *gl_pathc* and *gl_pathv* in *pglob* to reflect the paths already scanned. If GLOB_ERR is not set and either *errfunc* is a null pointer or (**errfunc*( )) returns 0, the error shall be ignored.

The set of error conditions that are considered to prevent *glob*( ) from expanding the pattern shall include [EACCES], [ENAMETOOLONG], and [ELOOP]. It is implementation-defined what other error conditions are included in the set.

The *glob*( ) function shall not fail because of large files.

RETURN VALUE

Upon successful completion, *glob*( ) shall return 0. The argument *pglob*−>*gl_pathc* shall return the number of matched pathnames and the argument *pglob*−>*gl_pathv* shall contain a pointer to a null-terminated list of matched and sorted pathnames. However, if *pglob*−>*gl_pathc* is 0, the content of *pglob*−>*gl_pathv* is undefined.

The *globfree*( ) function shall not return a value.

If *glob*( ) terminates due to an error, it shall return one of the non-zero constants defined in **<glob.h>**. The arguments *pglob*−>*gl_pathc* and *pglob*−>*gl_pathv* are still set as defined above.

ERRORS

The *glob*( ) function shall fail and return the corresponding value if:

GLOB_ABORTED — The scan was stopped because (**errfunc*( )) was called and returned non-zero, or, optionally, *errfunc* was a null pointer and an attempt to open, read, or search a directory failed because of an error condition that is not related to file system contents, or GLOB_ERR was set.

GLOB_NOMATCH — The pattern does not match any existing pathname, and GLOB_NOCHECK was not set in flags.

GLOB_NOSPACE — An attempt to allocate memory failed.

EXAMPLES

One use of the GLOB_DOOFFS flag is by applications that build an argument list for use with *execv*( ), *execve*( ), or *execvp*( ). Suppose, for example, that an application wants to do the equivalent of:

```
ls -ld -- *.c
```

but for some reason:

```
system("ls -ld -- *.c")
```

is not acceptable. The application could obtain the same result (except for error handling, omitted here for simplicity) using the sequence:

```
globbuf.gl_offs = 3;
glob("*.c", GLOB_DOOFFS|GLOB_NOCHECK, NULL, &globbuf);
globbuf.gl_pathv[0] = "ls";  // to establish the initial arguments
```



```
41250       globbuf.gl_pathv[1] = "-ld"; // that sh -c "ls -ld --" would
41251       globbuf.gl_pathv[2] = "--";  // produce for both examples
41252       execvp("ls", &globbuf.gl_pathv[0]);
```

Using the same example:

```
41254       ls -ld -- *.c *.h
```

could be simulated using GLOB_APPEND as follows:

```
41256       globbuf.gl_offs = 3;
41257       glob("*.c", GLOB_DOOFFS|GLOB_NOCHECK, NULL, &globbuf);
41258       glob("*.h", GLOB_DOOFFS|GLOB_NOCHECK|GLOB_APPEND, NULL, &globbuf);
41259       ...
```

APPLICATION USAGE

This function is not provided for the purpose of enabling utilities to perform pathname expansion on their arguments, as this operation is performed by the shell, and utilities are explicitly not expected to redo this. Instead, it is provided for applications that need to do pathname expansion on strings obtained from other sources, such as a pattern typed by a user or read from a file.

If a utility needs to see if a pathname matches a given pattern, it can use *fnmatch*( ).

Note that *gl_pathc* and *gl_pathv* have meaning even if *glob*( ) fails. This allows *glob*( ) to report partial results in the event of an error. However, if *gl_pathc* is 0, *gl_pathv* is unspecified even if *glob*( ) did not return an error.

The GLOB_NOCHECK option could be used when an application wants to expand a pathname if wildcards are specified, but wants to treat the pattern as just a string otherwise. The *sh* utility might use this for option-arguments, for example.

The new pathnames generated by a subsequent call with GLOB_APPEND are not sorted together with the previous pathnames. This mirrors the way that the shell handles pathname expansion when multiple expansions are done on a command line.

It is recommended that (**errfunc*( )) should always return a non-zero value if the *eerrno* parameter indicates an error condition that is not related to file system contents. See XRAT Section C.2.14.3 (on page 3911) for information about which error conditions are related to file system contents.

Applications that need tilde and parameter expansion should use *wordexp*( ).

RATIONALE

It was claimed that the GLOB_DOOFFS flag is unnecessary because it could be simulated using:

```
41283       new = (char **)malloc((n + pglob->gl_pathc + 1)
41284             * sizeof(char *));
41285       (void) memcpy(new+n, pglob->gl_pathv,
41286             pglob->gl_pathc * sizeof(char *));
41287       (void) memset(new, 0, n * sizeof(char *));
41288       free(pglob->gl_pathv);
41289       pglob->gl_pathv = new;
```

However, this assumes that the memory pointed to by *gl_pathv* is a block that was separately created using *malloc*( ). This is not necessarily the case. An application should make no assumptions about how the memory referenced by fields in *pglob* was allocated. It might have been obtained from *malloc*( ) in a large chunk and then carved up within *glob*( ), or it might have been created using a different memory allocator. It is not the intent of the standard developers to

specify or imply how the memory used by *glob*( ) is managed.

The GLOB_APPEND flag would be used when an application wants to expand several different patterns into a single list.

Earlier versions of this standard defined the behavior associated with the flag GLOB_MARK as: ``Each pathname that is a directory that matches *pattern* shall have a <slash> appended.'' This was undesirable if the matched pathname was <slash> or if the matched pathname was <slash><slash> and the implementation treats a leading <slash><slash> differently than it treats a pathname with a single leading <slash>. Only a few implementations were known to conform to this requirement (maybe only one) and there was a lot of variation in the way other implementations behaved. The current wording allows many of the alternative behaviors that were observed, except that the pathnames `"/"` and `"//"` (if it is treated differently than `"/"`) must not be modified.

Implementations should consider the following much simpler requirement (which is allowed by the current standard) when processing the GLOB_MARK flag: ``Each pathname that matches *pattern*, is determined to be a directory after pathname resolution, and does not end with a <slash> shall have a <slash> appended.''

Implementations differ as to which error conditions they consider prevent *glob*( ) from expanding the pattern. The standard requires that [EACCES], [ENAMETOOLONG], and [ELOOP] are included because in these cases the expansion could succeed if performed with a different effective user or group ID, or with an alternative pathname (shorter than {PATH_MAX}, or traversing fewer symbolic links).

Implementations are encouraged to call (**errfunc*( )) for all error conditions that are related to file system contents which occur when attempting to open or search a pathname as a directory or attempting to read an opened directory. The appropriate way to handle such errors varies according to the provenance of the pattern and what the application will do with the resulting pathnames, and should therefore be for the application to decide. For example, given the pattern `"non-existing/*"`, some applications may want *glob*( ) to succeed and return an empty list because there are no existing files that match the pattern, but for others that would not be appropriate, such as if an application asks the user to name a directory containing files to be processed and the user makes a typing mistake when responding; the application will want to alert the user to the mistake instead of behaving as if the user had named an empty directory. If (**errfunc*( )) is called for [ENOENT] errors, the first application can ignore them in that function, but if (**errfunc*( )) is not called, the second application cannot achieve what it wants using *glob*( ). Similar reasoning applies for the pattern `"regular_file/*"` and [ENOTDIR] errors.

FUTURE DIRECTIONS

A future version of this standard may require that (**errfunc*( )) is called for all error conditions that are related to file system contents which occur when attempting to open or search a pathname as a directory or attempting to read an opened directory.

SEE ALSO

exec, *fdopendir*( ), *fnmatch*( ), *fstatat*( ), *readdir*( ), *wordexp*( )

XBD Section 7.3.2 (on page 139), **<glob.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 5

Moved from POSIX2 C-language Binding to BASE.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

The **restrict** keyword is added to the *glob*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 385 is applied, adding a requirement that *globfree*( ) does not modify *errno* when passed a pointer to a **glob_t** that can be freed.

Austin Group Defect 1255 is applied, changing the EXAMPLES section.

Austin Group Defects 1273 and 1275 are applied, clarifying how errors are treated when attempting to open or search a pathname as a directory or attempting to read an opened directory.

Austin Group Defect 1300 is applied, changing the description of GLOB_MARK.

Austin Group Defect 1444 is applied, correcting cross-references to *wordexp*( ).

41353 **NAME**

41354 gmtime, gmtime_r — convert a time value to a broken-down UTC time

41355 **SYNOPSIS**

41356 ```
#include <time.h>
```

41357 ```
struct tm *gmtime(const time_t *timer);
```

41358 CX ```
struct tm *gmtime_r(const time_t *restrict timer,
```

41359 ```
    struct tm *restrict result);
```

41360 **DESCRIPTION**

41361 CX For *gmtime*( ): The functionality described on this reference page is aligned with the ISO C
41362 standard. Any conflict between the requirements described here and the ISO C standard is
41363 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

41364 The *gmtime*( ) function shall convert the time in seconds since the Epoch pointed to by *timer* into
41365 a broken-down time, expressed as Coordinated Universal Time (UTC).

41366 CX The relationship between a time in seconds since the Epoch used as an argument to *gmtime*( )
41367 and the **tm** structure (defined in the **<time.h>** header) is that the result shall be as specified in
41368 the expression given in the definition of seconds since the Epoch (see XBD Section 4.19, on page
41369 107), where the names in the structure and in the expression correspond.

41370 The same relationship shall apply for *gmtime_r*( ).

41371 The *gmtime*( ) function need not be thread-safe; however, *gmtime*( ) shall avoid data races with all
41372 functions other than itself, *asctime*( ), *ctime*( ), and *localtime*( ).

41373 The *asctime*( ), *ctime*( ), *gmtime*( ), and *localtime*( ) functions shall return values in one of two static
41374 objects: a broken-down time structure and an array of type **char**. Execution of any of the
41375 functions that return a pointer to one of these object types may overwrite the information in any
41376 object of the same type pointed to by the value returned from any previous call to any of them.

41377 CX The *gmtime_r*( ) function shall convert the time in seconds since the Epoch pointed to by *timer*
41378 into a broken-down time expressed as Coordinated Universal Time (UTC). The broken-down
41379 time is stored in the structure referred to by *result*. The *gmtime_r*( ) function shall also return the
41380 address of the same structure.

41381 **RETURN VALUE**

41382 Upon successful completion, the *gmtime*( ) function shall return a pointer to a **struct tm**. If an
41383 CX error is detected, *gmtime*( ) shall return a null pointer and set *errno* to indicate the error.

41384 Upon successful completion, *gmtime_r*( ) shall return the address of the structure pointed to by
41385 the argument *result*. The structure's *tm_zone* member shall be set to a pointer to the string
41386 `"UTC"`, which shall have static storage duration. If an error is detected, *gmtime_r*( ) shall return a
41387 null pointer and set *errno* to indicate the error.

41388 **ERRORS**

41389 CX The *gmtime*( ) and *gmtime_r*( ) functions shall fail if:

41390 CX [EOVERFLOW] The result cannot be represented.

EXAMPLES

None.

APPLICATION USAGE

The *gmtime_r*( ) function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime( ), *clock*( ), *ctime*( ), *difftime*( ), *futimens*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( )

XBD Section 4.19 (on page 107), **<time.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

A note indicating that the *gmtime*( ) function need not be reentrant is added to the DESCRIPTION.

The *gmtime_r*( ) function is included for alignment with the POSIX Threads Extension.

Issue 6

The *gmtime_r*( ) function is marked as part of the Thread-Safe Functions option.

Extensions beyond the ISO C standard are marked.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

The **restrict** keyword is added to the *gmtime_r*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/27 is applied, adding the [EOVERFLOW] error.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/48 is applied, updating the error handling for *gmtime_r*( ).

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

The *gmtime_r*( ) function is moved from the Thread-Safe Functions option to the Base.

Issue 8

Austin Group Defect 1302 is applied, aligning the *gmtime*( ) function with the ISO/IEC 9899: 2018 standard.

Austin Group Defect 1376 is applied, removing CX shading from some text derived from the ISO C standard and updating it to match the ISO C standard.

Austin Group Defect 1533 is applied, adding *tm_gmtoff* and *tm_zone* to the **tm** structure.

41429 **NAME**

41430 grantpt — grant access to the subsidiary pseudo-terminal device

41431 **SYNOPSIS**

41432 XSI
```
#include <stdlib.h>

int grantpt(int fildes);
```

41433

41434 **DESCRIPTION**

41435 The *grantpt*( ) function shall change the mode and ownership of the subsidiary pseudo-terminal
41436 device associated with its manager pseudo-terminal counterpart. The *fildes* argument is a file
41437 descriptor that refers to a manager pseudo-terminal device. The user ID of the subsidiary shall
41438 be set to the real UID of the calling process and the group ID shall be set to an unspecified group
41439 ID. The permission mode of the subsidiary pseudo-terminal shall be set to readable and writable
41440 by the owner, and writable by the group.

41441 The behavior of the *grantpt*( ) function is unspecified if the application has installed a signal
41442 handler to catch SIGCHLD signals.

41443 **RETURN VALUE**

41444 Upon successful completion, *grantpt*( ) shall return 0; otherwise, it shall return –1 and set *errno* to
41445 indicate the error.

41446 **ERRORS**

41447 The *grantpt*( ) function may fail if:

41448 [EACCES] The corresponding subsidiary pseudo-terminal device could not be accessed.

41449 [EBADF] The *fildes* argument is not a valid open file descriptor.

41450 [EINVAL] The *fildes* argument is not associated with a manager pseudo-terminal device.

41451 **EXAMPLES**

41452 None.

41453 **APPLICATION USAGE**

41454 None.

41455 **RATIONALE**

41456 See the RATIONALE section for *posix_openpt*( ).

41457 **FUTURE DIRECTIONS**

41458 None.

41459 **SEE ALSO**

41460 *open*( ), *posix_openpt*( ), *ptsname*( ), *unlockpt*( )

41461 XBD **<stdlib.h>**

41462 **CHANGE HISTORY**

41463 First released in Issue 4, Version 2.

41464 **Issue 5**

41465 Moved from X/OPEN UNIX extension to BASE.

41466 The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0272 [96] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0177 [506] is applied.

Issue 8

Austin Group Defect 1466 is applied, changing the terminology used for pseudo-terminal devices.

41473 **NAME**

41474 hcreate, hdestroy, hsearch — manage hash search table

41475 **SYNOPSIS**

41476 XSI
```
#include <search.h>

int hcreate(size_t nel);
void hdestroy(void);
ENTRY *hsearch(ENTRY item, ACTION action);
```

41480 **DESCRIPTION**

41481 The *hcreate*( ), *hdestroy*( ), and *hsearch*( ) functions shall manage hash search tables.

41482 The *hcreate*( ) function shall allocate sufficient space for the table, and the application shall
41483 ensure it is called before *hsearch*( ) is used. The *nel* argument is an estimate of the maximum
41484 number of entries that the table shall contain. This number may be adjusted upward by the
41485 algorithm in order to obtain certain mathematically favorable circumstances.

41486 The *hdestroy*( ) function shall dispose of the search table, and may be followed by another call to
41487 *hcreate*( ). After the call to *hdestroy*( ), the data can no longer be considered accessible.

41488 The *hsearch*( ) function is a hash-table search routine. It shall return a pointer into a hash table
41489 indicating the location at which an entry can be found. The *item* argument is a structure of type
41490 **ENTRY** (defined in the **<search.h>** header) containing two pointers: *item.key* points to the
41491 comparison key (a **char** *), and *item.data* (a **void** *) points to any other data to be associated with
41492 that key. The comparison function used by *hsearch*( ) is *strcmp*( ). The *action* argument is a
41493 member of an enumeration type **ACTION** indicating the disposition of the entry if it cannot be
41494 found in the table. ENTER indicates that the item should be inserted in the table at an
41495 appropriate point. FIND indicates that no entry should be made. Unsuccessful resolution is
41496 indicated by the return of a null pointer.

41497 These functions need not be thread-safe.

41498 **RETURN VALUE**

41499 The *hcreate*( ) function shall return 0 if it cannot allocate sufficient space for the table; otherwise,
41500 it shall return non-zero.

41501 The *hdestroy*( ) function shall not return a value.

41502 The *hsearch*( ) function shall return a null pointer if either the action is FIND and the item could
41503 not be found or the action is ENTER and the table is full.

41504 **ERRORS**

41505 The *hcreate*( ) and *hsearch*( ) functions may fail if:

41506 [ENOMEM] Insufficient storage space is available.

41507 **EXAMPLES**

41508 The following example reads in strings followed by two numbers and stores them in a hash
41509 table, discarding duplicates. It then reads in strings and finds the matching entry in the hash
41510 table and prints it out.

```
#include <stdio.h>
#include <search.h>
#include <string.h>

struct info {        /* This is the info stored in the table */
    int age, room;   /* other than the key. */
};
```



```
#define NUM_EMPL    5000    /* # of elements in search table. */
int main(void)
{
    char string_space[NUM_EMPL*20];   /* Space to store strings. */
    struct info info_space[NUM_EMPL]; /* Space to store employee info. */
    char *str_ptr = string_space;     /* Next space in string_space. */
    struct info *info_ptr = info_space;
                                      /* Next space in info_space. */
    ENTRY item;
    ENTRY *found_item; /* Name to look for in table. */
    char name_to_find[30];

    int i = 0;

    /* Create table; no error checking is performed. */
    (void) hcreate(NUM_EMPL);
    while (scanf("%s%d%d", str_ptr, &info_ptr->age,
           &info_ptr->room) != EOF && i++ < NUM_EMPL) {

        /* Put information in structure, and structure in item. */
        item.key = str_ptr;
        item.data = info_ptr;
        str_ptr += strlen(str_ptr) + 1;
        info_ptr++;

        /* Put item into table. */
        (void) hsearch(item, ENTER);
    }

    /* Access table. */
    item.key = name_to_find;
    while (scanf("%s", item.key) != EOF) {
        if ((found_item = hsearch(item, FIND)) != NULL) {

            /* If item is in the table. */
            (void)printf("found %s, age = %d, room = %d\n",
                found_item->key,
                ((struct info *)found_item->data)->age,
                ((struct info *)found_item->data)->room);
        } else
            (void)printf("no such employee %s\n", name_to_find);
    }
    return 0;
}
```

APPLICATION USAGE

The *hcreate*( ) and *hsearch*( ) functions may use *malloc*( ) to allocate space.

RATIONALE

None.

FUTURE DIRECTIONS

None.

41561 **SEE ALSO**

41562 *bsearch*( ), *lsearch*( ), *malloc*( ), *strcmp*( ), *tdelete*( )

41563 XBD **<search.h>**

41564 **CHANGE HISTORY**

41565 First released in Issue 1. Derived from Issue 1 of the SVID.

41566 **Issue 6**

41567 The normative text is updated to avoid use of the term ``must'' for application requirements.

41568 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

41569 **Issue 7**

41570 Austin Group Interpretation 1003.1-2001 #156 is applied.

41571 **NAME**

41572 htobe16, htobe32, htobe64, htole16, htole32, htole64 — convert values between host and
41573 specified byte order

41574 **SYNOPSIS**

41575 ```
#include <endian.h>
```

41576 ```
uint16_t htobe16(uint16_t host_16bits);
```
41577 ```
uint32_t htobe32(uint32_t host_32bits);
```
41578 ```
uint64_t htobe64(uint64_t host_64bits);
```
41579 ```
uint16_t htole16(uint16_t host_16bits);
```
41580 ```
uint32_t htole32(uint32_t host_32bits);
```
41581 ```
uint64_t htole64(uint64_t host_64bits);
```

41582 **DESCRIPTION**

41583 Refer to *be16toh( )*.

  
```

NAME

htonl, htons, ntohl, ntohs — convert values between host and network byte order

SYNOPSIS

```
#include <arpa/inet.h>

uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

DESCRIPTION

These functions shall convert 16-bit and 32-bit quantities between network byte order and host byte order.

On some implementations, these functions are defined as macros.

The **uint32_t** and **uint16_t** types are defined in **<inttypes.h>**.

RETURN VALUE

The *htonl*( ) and *htons*( ) functions shall return the argument value converted from host to network byte order.

The *ntohl*( ) and *ntohs*( ) functions shall return the argument value converted from network to host byte order.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

These functions are most often used in conjunction with IPv4 addresses and ports as returned by *gethostent*( ) and *getservent*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

be16toh( ), *endhostent*( ), *endservent*( )

XBD **<arpa/inet.h>**, **<endian.h>**, **<inttypes.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 8

Austin Group Defect 162 is applied, adding *be16toh*( ) and **<endian.h>** to the SEE ALSO section.

NAME

hypot, hypotf, hypotl — Euclidean distance function

SYNOPSIS

```
#include <math.h>

double hypot(double x, double y);
float hypotf(float x, float y);
long double hypotl(long double x, long double y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the value of the square root of x^2+y^2 without undue overflow or underflow.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the length of the hypotenuse of a right-angled triangle with sides of length *x* and *y*.

If the correct value would cause overflow, a range error shall occur and *hypot*( ), *hypotf*( ), and *hypotl*( ) shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

MX If *x* or *y* is ±Inf, +Inf shall be returned (even if one of *x* or *y* is NaN).

If *x* or *y* is NaN, and the other is not ±Inf, a NaN shall be returned.

MXX If both arguments are subnormal and the correct result is subnormal, a range error may occur and the correct result shall be returned.

ERRORS

These functions shall fail if:

Range Error — The result overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

MX Range Error — The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

See the EXAMPLES section in *atan2*( ).

APPLICATION USAGE

hypot(*x*,*y*), *hypot*(*y*,*x*), and *hypot*(*x*, –*y*) are equivalent.

hypot(*x*, ±0) is equivalent to *fabs*(*x*).

Underflow only happens when both *x* and *y* are subnormal and the (inexact) result is also subnormal.

These functions take precautions against overflow during intermediate steps of the computation.

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

atan2( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *sqrt*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *hypot*( ) function is no longer marked as an extension.

The *hypotf*( ) and *hypotl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/49 is applied, updating the EXAMPLES section.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0273 [68] is applied.

NAME

iconv — codeset conversion function

SYNOPSIS

```
#include <iconv.h>

size_t iconv(iconv_t cd, char **restrict inbuf,
    size_t *restrict inbytesleft, char **restrict outbuf,
    size_t *restrict outbytesleft);
```

DESCRIPTION

The *iconv*( ) function shall convert the sequence of characters from one codeset, in the array specified by *inbuf*, into a sequence of corresponding characters in another codeset, in the array specified by *outbuf*. The codesets are those specified in the *iconv_open*( ) call that returned the conversion descriptor, *cd*. The *inbuf* argument points to a variable that points to the first character in the input buffer and *inbytesleft* indicates the number of bytes to the end of the buffer to be converted. The *outbuf* argument points to a variable that points to the first available byte in the output buffer and *outbytesleft* indicates the number of the available bytes to the end of the buffer.

For state-dependent encodings, the conversion descriptor *cd* is placed into its initial shift state by a call for which *inbuf* is a null pointer, or for which *inbuf* points to a null pointer. When *iconv*( ) is called in this way, and if *outbuf* is not a null pointer or a pointer to a null pointer, and *outbytesleft* points to a positive value, *iconv*( ) shall place, into the output buffer, the byte sequence to change the output buffer to its initial shift state. If the output buffer is not large enough to hold the entire reset sequence, *iconv*( ) shall fail and set *errno* to [E2BIG]. Subsequent calls with *inbuf* as other than a null pointer or a pointer to a null pointer cause the conversion to take place from the current state of the conversion descriptor.

If a sequence of input bytes does not form a valid character or shift sequence in the input codeset:

- If the //IGNORE indicator suffix was specified when the conversion descriptor *cd* was opened and the byte sequence is immediately followed by a valid character or shift sequence, the sequence of bytes shall be discarded and conversion shall continue from the immediately following valid character or shift sequence. This shall not be treated as an error.
- If the //IGNORE indicator suffix was not specified when the conversion descriptor *cd* was opened, conversion shall stop after the previous successfully converted character or shift sequence.

If the input buffer ends with an incomplete character or shift sequence, conversion shall stop after the previous successfully converted bytes. If the output buffer is not large enough to hold the entire converted input, conversion shall stop just prior to the input bytes that would cause the output buffer to overflow. The variable pointed to by *inbuf* shall be updated to point to the byte following the last byte successfully used in the conversion. The value pointed to by *inbytesleft* shall be decremented to reflect the number of bytes still not converted in the input buffer. The variable pointed to by *outbuf* shall be updated to point to the byte following the last byte of converted output data. The value pointed to by *outbytesleft* shall be decremented to reflect the number of bytes still available in the output buffer. For state-dependent encodings, the conversion descriptor shall be updated to reflect the shift state in effect at the end of the last successfully converted byte sequence.

If *iconv*( ) encounters a character in the input buffer that is valid, but for which an identical character does not exist in the output codeset:

- If either the //IGNORE or the //NON_IDENTICAL_DISCARD indicator suffix was specified when the conversion descriptor *cd* was opened, the character shall be discarded but shall still be counted in the return value of the *iconv*( ) call.
- If the //TRANSLIT indicator suffix was specified when the conversion descriptor *cd* was opened, an implementation-defined transliteration shall be performed, if possible, to convert the character into one or more characters of the output codeset that best resemble the input character. The character shall be counted as one character in the return value of the *iconv*( ) call, regardless of the number of output characters.
- If no indicator suffix was specified when the conversion descriptor *cd* was opened, or the //TRANSLIT indicator suffix was specified but no transliteration of the character is possible, *iconv*( ) shall perform an implementation-defined conversion on the character and it shall be counted in the return value of the *iconv*( ) call.

RETURN VALUE

The *iconv*( ) function shall update the variables pointed to by the arguments to reflect the extent of the conversion and shall return the number of input characters that could not be converted to an identical output character. If the entire string in the input buffer is converted, except for any byte sequences discarded as a result of the //IGNORE indicator suffix, the value pointed to by *inbytesleft* shall be 0. If the input conversion is stopped due to any conditions mentioned above, the value pointed to by *inbytesleft* shall be non-zero and *errno* shall be set to indicate the condition. If an error occurs, *iconv*( ) shall return (**size_t**)–1 and set *errno* to indicate the error.

ERRORS

The *iconv*( ) function shall fail if:

[EILSEQ] Input conversion stopped due to an input byte that does not belong to the input codeset.

[E2BIG] Input conversion stopped due to lack of space in the output buffer.

[EINVAL] Input conversion stopped due to an incomplete character or shift sequence at the end of the input buffer.

The *iconv*( ) function may fail if:

[EBADF] The *cd* argument is not a valid open conversion descriptor.

EXAMPLES

None.

APPLICATION USAGE

The *inbuf* argument indirectly points to the memory area which contains the conversion input data. The *outbuf* argument indirectly points to the memory area which is to contain the result of the conversion. The objects indirectly pointed to by *inbuf* and *outbuf* are not restricted to containing data that is directly representable in the ISO C standard language **char** data type. The type of *inbuf* and *outbuf*, **char** **, does not imply that the objects pointed to are interpreted as null-terminated C strings or arrays of characters. Any interpretation of a byte sequence that represents a character in a given character set encoding scheme is done internally within the codeset converters. For example, the area pointed to indirectly by *inbuf* and/or *outbuf* can contain all zero octets that are not interpreted as string terminators but as coded character data according to the respective codeset encoding scheme. The type of the data (**char**, **short**, **long**, and so on) read or stored in the objects is not specified, but may be inferred for both the input and output data by the converters determined by the *fromcode* and *tocode* arguments of *iconv_open*( ).

Regardless of the data type inferred by the converter, the size of the remaining space in both input and output objects (the *intbytesleft* and *outbytesleft* arguments) is always measured in bytes.

For implementations that support the conversion of state-dependent encodings, the conversion descriptor must be able to accurately reflect the shift-state in effect at the end of the last successful conversion. It is not required that the conversion descriptor itself be updated, which would require it to be a pointer type. Thus, implementations are free to implement the descriptor as a handle (other than a pointer type) by which the conversion information can be accessed and updated.

It is the responsibility of the application to ensure that, if the output codeset has a locking-shift encoding, the output buffer is returned to its initial shift state when conversion is completed. This can be accomplished by calling *iconv*( ) with *inbuf* as a null pointer, or with *inbuf* pointing to a null pointer, before calling *iconv_close*( ). Since the standard does not provide a way to query whether a codeset has a locking-shift encoding, it is recommended that applications always call *iconv*( ) in this way before calling *iconv_close*( ).

When the //IGNORE indicator suffix was used to open the conversion descriptor, *iconv*( ) does not provide any indication of whether any invalid input byte sequences were discarded. Applications which need to detect the discarding of invalid input byte sequences can open the conversion descriptor without using //IGNORE and then call *iconv*( ) in a loop such that if it returns an [EILSEQ] error, the application increments the variable pointed to by *inbuf* and decrements the variable pointed to by *inbytesleft* before the next call. This technique can also be used by applications which need to use //TRANSLIT but also discard invalid input byte sequences.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iconv_open( ), *iconv_close*( ), *mbsrtowcs*( )

XBD **<iconv.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the HP-UX Manual.

Issue 6

The SYNOPSIS has been corrected to align with the **<iconv.h>** reference page.

The **restrict** keyword is added to the *iconv*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

The *iconv*( ) function is moved from the XSI option to the Base.

Issue 8

Austin Group Defect 1007 is applied, adding support for indicator suffixes in the *tocode* argument to *iconv_open*( ).

Austin Group Defect 1008 is applied, adding a paragraph about locking-shift encodings to the APPLICATION USAGE section.

Austin Group Defect 1438 is applied, changing ``valid character in the specified codeset'' to ``valid character in the specified input codeset''.

41829 **NAME**

41830 iconv_close — codeset conversion deallocation function

41831 **SYNOPSIS**

41832 ```
#include <iconv.h>
```

41833 ```
int iconv_close(iconv_t cd);
```

41834 **DESCRIPTION**

41835 The *iconv_close*( ) function shall deallocate the conversion descriptor *cd* and all other associated
41836 resources allocated by *iconv_open*( ).

41837 If a file descriptor is used to implement the type **iconv_t**, that file descriptor shall be closed.

41838 **RETURN VALUE**

41839 Upon successful completion, 0 shall be returned; otherwise, –1 shall be returned and *errno* set to
41840 indicate the error.

41841 **ERRORS**

41842 The *iconv_close*( ) function may fail if:

41843 [EBADF] The conversion descriptor is invalid.

41844 **EXAMPLES**

41845 None.

41846 **APPLICATION USAGE**

41847 None.

41848 **RATIONALE**

41849 None.

41850 **FUTURE DIRECTIONS**

41851 None.

41852 **SEE ALSO**

41853 *iconv*( ), *iconv_open*( )

41854 XBD **<iconv.h>**

41855 **CHANGE HISTORY**

41856 First released in Issue 4. Derived from the HP-UX Manual.

41857 **Issue 7**

41858 The *iconv_close*( ) function is moved from the XSI option to the Base.

41859 **NAME**

41860 iconv_open — codeset conversion allocation function

41861 **SYNOPSIS**

41862 `#include <iconv.h>`

41863 `iconv_t iconv_open(const char *tocode, const char *fromcode);`

41864 **DESCRIPTION**

41865 The *iconv_open*( ) function shall return a conversion descriptor that describes a conversion from
41866 the codeset specified by the string pointed to by the *fromcode* argument to the codeset specified
41867 by the string pointed to by the *tocode* argument. For state-dependent encodings, the conversion
41868 descriptor shall be in a codeset-dependent initial shift state, ready for immediate use with
41869 *iconv*( ).

41870 The codeset names that can be specified in *fromcode* and *tocode* and their permitted combinations
41871 are implementation-defined. Any one of the following indicator suffixes can be appended to the
41872 codeset name in *tocode*:

41873 //IGNORE Discard input bytes that do not form a valid character or shift sequence, and
41874 discard input characters for which an identical character does not exist in the
41875 output codeset.

41876 //NON_IDENTICAL_DISCARD
41877 Discard input characters for which an identical character does not exist in the
41878 output codeset.

41879 //TRANSLIT Transliterate input characters for which an identical character does not exist in
41880 the output codeset into one or more characters of the output codeset that best
41881 resemble the input character.

41882 See the description of *iconv*( ) for details of how these indicator suffixes alter the conversion
41883 performed by *iconv*( ). Additional implementation-defined indicator suffixes may be supported.

41884 A conversion descriptor shall remain valid until it is closed by *iconv_close*( ) or an implicit close.

41885 If a file descriptor is used to implement conversion descriptors, the FD_CLOEXEC flag shall be
41886 set; see **<fcntl.h>**.

41887 **RETURN VALUE**

41888 Upon successful completion, *iconv_open*( ) shall return a conversion descriptor for use on
41889 subsequent calls to *iconv*( ). Otherwise, *iconv_open*( ) shall return (**iconv_t**)–1 and set *errno* to
41890 indicate the error.

41891 **ERRORS**

41892 The *iconv_open*( ) function may fail if:

41893 [EMFILE] All file descriptors available to the process are currently open.

41894 [ENFILE] Too many files are currently open in the system.

41895 [ENOMEM] Insufficient storage space is available.

41896 [EINVAL] Conversion from the codeset specified in *fromcode* to the codeset specified in
41897 *tocode* is not supported by the implementation, or the codeset name in *tocode* is
41898 followed by an indicator suffix that is unrecognized or not supported.

EXAMPLES

None.

APPLICATION USAGE

Some implementations of *iconv_open*( ) use *malloc*( ) to allocate space for internal buffer areas. The *iconv_open*( ) function may fail if there is insufficient storage space to accommodate these buffers.

Conforming applications must assume that conversion descriptors are not valid after a call to one of the *exec* functions.

Application developers should consult the system documentation to determine the supported codesets and their naming schemes.

Some implementations of *iconv_open*( ) allow appending multiple indicator suffixes to the codeset name in *tocode*, and some allow appending an indicator suffix (or suffixes) in both *fromcode* and *tocode*. Portable applications should append at most one indicator suffix, and append it only in *tocode*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iconv( ), *iconv_close*( )

XBD **<fcntl.h>**, **<iconv.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the HP-UX Manual.

Issue 7

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

The *iconv_open*( ) function is moved from the XSI option to the Base.

Issue 8

Austin Group Defect 1007 is applied, adding support for indicator suffixes in the *tocode* argument to *iconv_open*( ).

41928 **NAME**

41929 if_freenameindex — free memory allocated by if_nameindex

41930 **SYNOPSIS**

41931 ```
#include <net/if.h>
```

41932 ```
void if_freenameindex(struct if_nameindex *ptr);
```

41933 **DESCRIPTION**

41934 The *if_freenameindex*( ) function shall free the memory allocated by *if_nameindex*( ). The *ptr*
41935 argument shall be a pointer that was returned by *if_nameindex*( ). After *if_freenameindex*( ) has
41936 been called, the application shall not use the array of which *ptr* is the address. The
41937 *if_freenameindex*( ) function shall not modify *errno* if *ptr* was previously returned by
41938 *if_nameindex*( ) and not yet freed.

41939 **RETURN VALUE**

41940 None.

41941 **ERRORS**

41942 No errors are defined.

41943 **EXAMPLES**

41944 None.

41945 **APPLICATION USAGE**

41946 None.

41947 **RATIONALE**

41948 None.

41949 **FUTURE DIRECTIONS**

41950 None.

41951 **SEE ALSO**

41952 *getsockopt*( ), *if_indextoname*( ), *if_nameindex*( ), *if_nametoindex*( ), *setsockopt*( )

41953 XBD **<net/if.h>**

41954 **CHANGE HISTORY**

41955 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

41956 **Issue 8**

41957 Austin Group Defect 385 is applied, adding a requirement that *if_freenameindex*( ) does not
41958 modify *errno* when passed a pointer to an **if_nameindex** structure than can be freed.

NAME

if_indextoname — map a network interface index to its corresponding name

SYNOPSIS

```
#include <net/if.h>

char *if_indextoname(unsigned ifindex, char *ifname);
```

DESCRIPTION

The *if_indextoname*( ) function shall map an interface index to its corresponding name.

When this function is called, *ifname* shall point to a buffer of at least {IF_NAMESIZE} bytes. The function shall place in this buffer the name of the interface with index *ifindex*.

RETURN VALUE

If *ifindex* is an interface index, then the function shall return the value supplied in *ifname*, which points to a buffer now containing the interface name. Otherwise, the function shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *if_indextoname*( ) function shall fail if:

[ENXIO] The interface does not exist.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getsockopt( ), *if_freenameindex*( ), *if_nameindex*( ), *if_nametoindex*( ), *setsockopt*( )

XBD **<net/if.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/28 is applied, changing {IFNAMSIZ} to {IF_NAMESIZ} in the DESCRIPTION.

41990 **NAME**

41991 if_nameindex — return all network interface names and indexes

41992 **SYNOPSIS**

41993 ```
#include <net/if.h>
```

41994 ```
struct if_nameindex *if_nameindex(void);
```

41995 **DESCRIPTION**

41996 The *if_nameindex*( ) function shall return an array of *if_nameindex* structures, one structure per
41997 interface. The end of the array is indicated by a structure with an *if_index* field of zero and an
41998 *if_name* field of NULL.

41999 Applications should call *if_freenameindex*( ) to release the memory that may be dynamically
42000 allocated by this function, after they have finished using it.

42001 **RETURN VALUE**

42002 An array of structures identifying local interfaces. A null pointer is returned upon an error, with
42003 *errno* set to indicate the error.

42004 **ERRORS**

42005 The *if_nameindex*( ) function may fail if:

42006 [ENOBUFS] Insufficient resources are available to complete the function.

42007 **EXAMPLES**

42008 None.

42009 **APPLICATION USAGE**

42010 None.

42011 **RATIONALE**

42012 None.

42013 **FUTURE DIRECTIONS**

42014 None.

42015 **SEE ALSO**

42016 *getsockopt*( ), *if_freenameindex*( ), *if_indextoname*( ), *if_nametoindex*( ), *setsockopt*( )

42017 XBD **<net/if.h>**

42018 **CHANGE HISTORY**

42019 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

NAME

if_nametoindex — map a network interface name to its corresponding index

SYNOPSIS

```
#include <net/if.h>

unsigned if_nametoindex(const char *ifname);
```

DESCRIPTION

The *if_nametoindex*( ) function shall return the interface index corresponding to name *ifname*.

RETURN VALUE

The corresponding index if *ifname* is the name of an interface; otherwise, zero.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getsockopt( ), *if_freenameindex*( ), *if_indextoname*( ), *if_nameindex*( ), *setsockopt*( )

XBD **<net/if.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

NAME

ilogb, ilogbf, ilogbl — return an unbiased exponent

SYNOPSIS

```
#include <math.h>

int ilogb(double x);
int ilogbf(float x);
int ilogbl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall return the exponent part of their argument x. Formally, the return value is the integral part of $\log_r |x|$ as a signed integral value, for non-zero x, where r is the radix of the machine's floating-point arithmetic, which is the value of FLT_RADIX defined in **<float.h>**.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the exponent part of x as a signed integer value. They are equivalent to calling the corresponding *logb*( ) function and casting the returned value to type **int**.

MX When the correct result is representable in the range of the return type, the returned value shall be exact and shall be independent of the current rounding direction mode.

XSI|MX If x is 0, the value FP_ILOGB0 shall be returned. On XSI-conformant systems and on systems that support the IEC 60559 Floating-Point option, a domain error shall occur; otherwise, a
CX domain error may occur.

XSI|MX If x is ±Inf, the value {INT_MAX} shall be returned. On XSI-conformant systems and on systems that support the IEC 60559 Floating-Point option, a domain error shall occur; otherwise,
CX a domain error may occur.

XSI|MX If x is a NaN, the value FP_ILOGBNAN shall be returned. On XSI-conformant systems and on systems that support the IEC 60559 Floating-Point option, a domain error shall occur; otherwise,
CX a domain error may occur.

If the correct value is greater than {INT_MAX} or less than {INT_MIN}, an unspecified value
XSI shall be returned. On XSI-conformant systems, a domain error shall occur and {INT_MAX} or {INT_MIN}, respectively, shall be returned;
MX if the IEC 60559 Floating-Point option is supported, a domain error shall occur; otherwise, a domain error or range error may occur.

ERRORS

These functions shall fail if:

XSI|MX **Domain Error** The correct value is not representable as an integer.

The x argument is zero, NaN, or ±Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception

shall be raised.

These functions may fail if:

Domain Error — The *x* argument is zero, NaN, or ±Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

The errors come from taking the expected floating-point value and converting it to **int**, which is an invalid operation in IEEE Std 754-1985 (since overflow, infinity, and NaN are not representable in a type **int**), so should be a domain error.

There are no known implementations that overflow. For overflow to happen, {INT_MAX} must be less than LDBL_MAX_EXP**log2*(FLT_RADIX) or {INT_MIN} must be greater than LDBL_MIN_EXP**log2*(FLT_RADIX) if subnormals are not supported, or {INT_MIN} must be greater than (LDBL_MIN_EXP-LDBL_MANT_DIG)**log2*(FLT_RADIX) if subnormals are supported.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *logb*( ), *scalbln*( )

XBD Section 4.23 (on page 109), **<float.h>**, **<math.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The *ilogb*( ) function is no longer marked as an extension.

The *ilogbf*( ) and *ilogbl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The RETURN VALUE section is revised for alignment with the ISO/IEC 9899:1999 standard.

Functionality relating to the XSI option is marked.

Issue 7

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #48 (SD5-XSH-ERN-71), #49, and #79 (SD5-XSH-ERN-72) are applied.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

NAME

imaxabs — return absolute value

SYNOPSIS

```
#include <inttypes.h>

intmax_t imaxabs(intmax_t j);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *imaxabs*( ) function shall compute the absolute value of an integer *j*. If the result cannot be represented, the behavior is undefined.

RETURN VALUE

The *imaxabs*( ) function shall return the absolute value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Since POSIX.1 requires a two's complement representation of **intmax_t**, the absolute value of the negative integer with the largest magnitude {INTMAX_MIN} is not representable, thus *imaxabs*(INTMAX_MIN) is undefined.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

imaxdiv( )

XBD **<inttypes.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 1108 is applied, changing the APPLICATION USAGE section.

42163 **NAME**

42164 imaxdiv — return quotient and remainder

42165 **SYNOPSIS**

42166 `#include <inttypes.h>`

42167 `imaxdiv_t imaxdiv(intmax_t numer, intmax_t denom);`

42168 **DESCRIPTION**

42169 CX The functionality described on this reference page is aligned with the ISO C standard. Any
42170 conflict between the requirements described here and the ISO C standard is unintentional. This
42171 volume of POSIX.1-2024 defers to the ISO C standard.

42172 The *imaxdiv*( ) function shall compute *numer* / *denom* and *numer* % *denom* in a single operation.

42173 **RETURN VALUE**

42174 The *imaxdiv*( ) function shall return a structure of type **imaxdiv_t**, comprising both the quotient
42175 and the remainder. The structure shall contain (in either order) the members *quot* (the quotient)
42176 and *rem* (the remainder), each of which has type **intmax_t**.

42177 If either part of the result cannot be represented, the behavior is undefined.

42178 **ERRORS**

42179 No errors are defined.

42180 **EXAMPLES**

42181 None.

42182 **APPLICATION USAGE**

42183 None.

42184 **RATIONALE**

42185 None.

42186 **FUTURE DIRECTIONS**

42187 None.

42188 **SEE ALSO**

42189 *imaxabs*( )

42190 XBD **<inttypes.h>**

42191 **CHANGE HISTORY**

42192 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

NAME

in6addr_any, in6addr_loopback — IPv6 address variables

SYNOPSIS

IP6

```
#include <netinet/in.h>

const struct in6_addr in6addr_any;
const struct in6_addr in6addr_loopback;
```

DESCRIPTION

The *in6addr_any* variable is initialized by the system to contain the wildcard IPv6 address (`::`).

The *in6addr_loopback* variable is initialized by the system to contain the loopback IPv6 address (`::1`).

RETURN VALUE

None.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

These variables were only described on the XBD **<netinet/in.h>** page in earlier versions of this standard.

FUTURE DIRECTIONS

None.

SEE ALSO

bind( ), *connect*( )

XBD **<netinet/in.h>**

CHANGE HISTORY

First released in Issue 8. Derived from Issue 7 XBD **<netinet/in.h>**.

NAME

inet_addr, inet_ntoa — IPv4 address manipulation

SYNOPSIS

OB

```
#include <arpa/inet.h>

in_addr_t inet_addr(const char *cp);
char *inet_ntoa(struct in_addr in);
```

DESCRIPTION

The *inet_addr*( ) function shall convert the string pointed to by *cp*, in the standard IPv4 dotted decimal notation, to an integer value suitable for use as an Internet address.

The *inet_ntoa*( ) function shall convert the Internet host address specified by *in* to a string in the Internet standard dot notation.

The *inet_ntoa*( ) function need not be thread-safe.

All Internet addresses shall be returned in network order (bytes ordered from left to right).

Values specified using IPv4 dotted decimal notation take one of the following forms:

`a.b.c.d` When four parts are specified, each shall be interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address.

`a.b.c` When a three-part address is specified, the last part shall be interpreted as a 16-bit quantity and placed in the rightmost two bytes of the network address. This makes the three-part address format convenient for specifying Class B network addresses as `"128.net.host"`.

`a.b` When a two-part address is supplied, the last part shall be interpreted as a 24-bit quantity and placed in the rightmost three bytes of the network address. This makes the two-part address format convenient for specifying Class A network addresses as `"net.host"`.

`a` When only one part is given, the value shall be stored directly in the network address without any byte rearrangement.

All numbers supplied as parts in IPv4 dotted decimal notation may be decimal, octal, or hexadecimal, as specified in the ISO C standard (that is, a leading 0x or 0X implies hexadecimal; otherwise, a leading `'0'` implies octal; otherwise, the number is interpreted as decimal).

RETURN VALUE

Upon successful completion, *inet_addr*( ) shall return the Internet address. Otherwise, it shall return (**in_addr_t**)(–1).

The *inet_ntoa*( ) function shall return a pointer to the network address in Internet standard dot notation.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The return value of *inet_ntoa*( ) may point to static data that may be overwritten by subsequent calls to *inet_ntoa*( ).

Applications should prefer *inet_pton*( ) over *inet_addr*( ) for the following reasons:

- The return value from *inet_addr*( ) when converting 255.255.255.255 is indistinguishable from an error.
- The *inet_pton*( ) function supports multiple address families.
- The alternative textual representations supported by *inet_addr*( ) (but not by *inet_pton*( )) are often used maliciously to confuse or mislead users (e.g., for phishing).

Applications should prefer *inet_ntop*( ) over *inet_ntoa*( ) as it supports multiple address families and is thread-safe.

RATIONALE

None.

FUTURE DIRECTIONS

These functions are included only for compatibility with older implementations and may be removed in a future version.

SEE ALSO

endhostent( ), *endnetent*( ), *inet_ntop*( )

XBD **<arpa/inet.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

Issue 8

Austin Group Defects 1101 and 1102 are applied, marking *inet_addr*( ) and *inet_ntoa*( ) as obsolescent.

NAME

inet_ntop, inet_pton — convert IPv4 and IPv6 addresses between binary and text form

SYNOPSIS

```
#include <arpa/inet.h>

const char *inet_ntop(int af, const void *restrict src,
    char *restrict dst, socklen_t size);
int inet_pton(int af, const char *restrict src, void *restrict dst);
```

DESCRIPTION

The *inet_ntop*( ) function shall convert a numeric address into a text string suitable for presentation. The *af* argument shall specify the family of the address. This can be AF_INET or AF_INET6. The *src* argument points to a buffer holding an IPv4 address if the *af* argument is AF_INET, or an IPv6 address if the *af* argument is AF_INET6; the address needs to be in network byte order. The *dst* argument points to a buffer where the function stores the resulting text string; it shall not be NULL. The *size* argument specifies the size of this buffer, which shall be large enough to hold the text string (INET_ADDRSTRLEN characters for IPv4, INET6_ADDRSTRLEN characters for IPv6).

The *inet_pton*( ) function shall convert an address in its standard text presentation form into its numeric binary form. The *af* argument shall specify the family of the address. The AF_INET and AF_INET6 address families shall be supported. The *src* argument points to the string being passed in. The *dst* argument points to a buffer into which the function stores the numeric address; this shall be large enough to hold the numeric address (32 bits for AF_INET, 128 bits for AF_INET6).

If the *af* argument of *inet_pton*( ) is AF_INET, the *src* string shall be in the standard IPv4 dotted-decimal form:

```
ddd.ddd.ddd.ddd
```

where `"ddd"` is a one to three digit decimal number between 0 and 255. Leading zeros shall be allowed. The *inet_pton*( ) function does not accept other formats (such as the octal numbers, hexadecimal numbers, and fewer than four numbers that *inet_addr*( ) accepts).

If the *af* argument of *inet_pton*( ) is AF_INET6, the *src* string shall be in one of the following standard IPv6 text forms:

1. The preferred form is `"x:x:x:x:x:x:x:x"`, where the `'x'`s are the hexadecimal values of the eight 16-bit pieces of the address. Leading zeros in individual fields can be omitted, but there shall be one to four hexadecimal digits in every field.
2. A string of contiguous zero fields in the preferred form can be shown as `"::"`. The `"::"` can only appear once in an address. Unspecified addresses (`"0:0:0:0:0:0:0:0"`) may be represented simply as `"::"`.
3. A third form that is sometimes more convenient when dealing with a mixed environment of IPv4 and IPv6 nodes is `"x:x:x:x:x:x:d.d.d.d"`, where the `'x'`s are the hexadecimal values of the six high-order 16-bit pieces of the address, and the `'d'`s are the decimal values of the four low-order 8-bit pieces of the address (standard IPv4 representation).

Note: A more extensive description of the standard representations of IPv6 addresses can be found in RFC 4291.

RETURN VALUE

The *inet_ntop*( ) function shall return a pointer to the buffer containing the text string if the conversion succeeds, and NULL otherwise, and set *errno* to indicate the error.

The *inet_pton*( ) function shall return 1 if the conversion succeeds, with the address pointed to by *dst* in network byte order. It shall return 0 if the input is not a valid IPv4 dotted-decimal string or a valid IPv6 address string, or −1 with *errno* set to [EAFNOSUPPORT] if the *af* argument is unknown.

IP6

ERRORS

The *inet_ntop*( ) and *inet_pton*( ) functions shall fail if:

[EAFNOSUPPORT]
The *af* argument is invalid.

[ENOSPC] The size of the *inet_ntop*( ) result buffer is inadequate.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

inet_addr( )

XBD **<arpa/inet.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

IPv6 extensions are marked.

The **restrict** keyword is added to the *inet_ntop*( ) and *inet_pton*( ) prototypes for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/29 is applied, adding ``the address must be in network byte order'' to the end of the fourth sentence of the first paragraph in the DESCRIPTION.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0178 [777] is applied.

Issue 8

Austin Group Defect 1102 is applied, removing a reference to the *inet_addr*( ) page from the DESCRIPTION.

Austin Group Defect 1573 is applied, clarifying that leading zeros are allowed in the IPv4 dotted-decimal form accepted by *inet_pton*( ).

Austin Group Defect 1685 is applied, updating RFC references.

NAME

initstate, random, setstate, srandom — pseudo-random number functions

SYNOPSIS

XSI

```
#include <stdlib.h>

char *initstate(unsigned seed, char *state, size_t size);
long random(void);
char *setstate(char *state);
void srandom(unsigned seed);
```

DESCRIPTION

The *random*( ) function shall use a non-linear additive feedback random-number generator employing a default state array size of 31 **long** integers to return successive pseudo-random numbers in the range from 0 to 2^{31}–1. The period of this random-number generator is approximately 16 x (2^{31}–1). The size of the state array determines the period of the random-number generator. Increasing the state array size shall increase the period.

With 256 bytes of state information, the period of the random-number generator shall be greater than 2^{69}.

Like *rand*( ), *random*( ) shall produce by default a sequence of numbers that can be duplicated by calling *srandom*( ) with 1 as the seed.

The *srandom*( ) function shall initialize the current state array using the value of *seed*.

The *initstate*( ) and *setstate*( ) functions handle restarting and changing random-number generators. The *initstate*( ) function allows a state array, pointed to by the *state* argument, to be initialized for future use. The *size* argument, which specifies the size in bytes of the state array, shall be used by *initstate*( ) to decide what type of random-number generator to use; the larger the state array, the more random the numbers. Values for the amount of state information are 8, 32, 64, 128, and 256 bytes. Other values greater than 8 bytes are rounded down to the nearest one of these values. If *initstate*( ) is called with $8 \le size < 32$, then *random*( ) shall use a simple linear congruential random number generator. The *seed* argument specifies a starting point for the random-number sequence and provides for restarting at the same point. The *initstate*( ) function shall return a pointer to the previous state information array.

If *initstate*( ) has not been called, then *random*( ) shall behave as though *initstate*( ) had been called with *seed*=1 and *size*=128.

Once a state has been initialized, *setstate*( ) allows switching between state arrays. The array defined by the *state* argument shall be used for further random-number generation until *initstate*( ) is called or *setstate*( ) is called again. The *setstate*( ) function shall return a pointer to the previous state array.

RETURN VALUE

If *initstate*( ) is called with *size* less than 8, it shall return NULL.

The *random*( ) function shall return the generated pseudo-random number.

The *srandom*( ) function shall not return a value.

Upon successful completion, *initstate*( ) and *setstate*( ) shall return a pointer to the previous state array; otherwise, a null pointer shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

After initialization, a state array can be restarted at a different point in one of two ways:

1. The *initstate*( ) function can be used, with the desired seed, state array, and size of the array.
2. The *setstate*( ) function, with the desired state, can be used, followed by *srandom*( ) with the desired seed. The advantage of using both of these functions is that the size of the state array does not have to be saved once it is initialized.

Although some implementations of *random*( ) have written messages to standard error, such implementations do not conform to POSIX.1-2024.

Issue 5 restored the historical behavior of this function.

Threaded applications should use *erand48*( ), *nrand48*( ), or *jrand48*( ) instead of *random*( ) when an independent random number sequence in multiple threads is required.

These functions should be avoided whenever non-trivial requirements (including safety) have to be fulfilled, unless seeded using *getentropy*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

drand48( ), *getentropy*( ), *rand*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

In the DESCRIPTION, the phrase ``values smaller than 8'' is replaced with ``values greater than or equal to 8, or less than 32'', ``*size*<8'' is replaced with ``8≤*size* <32'', and a new first paragraph is added to the RETURN VALUE section. A note is added to the APPLICATION USAGE indicating that these changes restore the historical behavior of the function.

Issue 6

In the DESCRIPTION, duplicate text ``For values greater than or equal to 8 . . .'' is removed.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/30 is applied, removing *rand_r*( ) from the list of suggested functions in the APPLICATION USAGE section.

Issue 7

The type of the first argument to *setstate*( ) is changed from **const char *** to **char ***.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0179 [743] is applied.

42448 **Issue 8**
42449 Austin Group Defect 1134 is applied, adding *getentropy*( ).

NAME

insque, remque — insert or remove an element in a queue

SYNOPSIS

XSI
```
#include <search.h>

void insque(void *element, void *pred);
void remque(void *element);
```

DESCRIPTION

The *insque*( ) and *remque*( ) functions shall manipulate queues built from doubly-linked lists. The queue can be either circular or linear. An application using *insque*( ) or *remque*( ) shall ensure it defines a structure in which the first two members of the structure are pointers to the same type of structure, and any further members are application-specific. The first member of the structure is a forward pointer to the next entry in the queue. The second member is a backward pointer to the previous entry in the queue. If the queue is linear, the queue is terminated with null pointers. The names of the structure and of the pointer members are not subject to any special restriction.

The *insque*( ) function shall insert the element pointed to by *element* into a queue immediately after the element pointed to by *pred*.

The *remque*( ) function shall remove the element pointed to by *element* from a queue.

If the queue is to be used as a linear list, invoking *insque*(&*element*, NULL), where *element* is the initial element of the queue, shall initialize the forward and backward pointers of *element* to null pointers.

If the queue is to be used as a circular list, the application shall ensure it initializes the forward pointer and the backward pointer of the initial element of the queue to the element's own address.

RETURN VALUE

The *insque*( ) and *remque*( ) functions do not return a value.

ERRORS

No errors are defined.

EXAMPLES

Creating a Linear Linked List

The following example creates a linear linked list.

```
#include <search.h>
...
struct myque element1;
struct myque element2;

char *data1 = "DATA1";
char *data2 = "DATA2";
...
element1.data = data1;
element2.data = data2;

insque (&element1, NULL);
insque (&element2, &element1);
```


Creating a Circular Linked List

The following example creates a circular linked list.

```
#include <search.h>
...
struct myque element1;
struct myque element2;

char *data1 = "DATA1";
char *data2 = "DATA2";
...
element1.data = data1;
element2.data = data2;

element1.fwd = &element1;
element1.bck = &element1;

insque (&element2, &element1);
```

Removing an Element

The following example removes the element pointed to by *element1*.

```
#include <search.h>
...
struct myque element1;
...
remque (&element1);
```

APPLICATION USAGE

The historical implementations of these functions described the arguments as being of type **struct qelem** * rather than as being of type **void** * as defined here. In those implementations, **struct qelem** was commonly defined in **<search.h>** as:

```
struct qelem {
    struct qelem  *q_forw;
    struct qelem  *q_back;
};
```

Applications using these functions, however, were never able to use this structure directly since it provided no room for the actual data contained in the elements. Most applications defined structures that contained the two pointers as the initial elements and also provided space for, or pointers to, the object's data. Applications that used these functions to update more than one type of table also had the problem of specifying two or more different structures with the same name, if they literally used **struct qelem** as specified.

As described here, the implementations were actually expecting a structure type where the first two members were forward and backward pointers to structures. With C compilers that didn't provide function prototypes, applications used structures as specified in the DESCRIPTION above and the compiler did what the application expected.

If this method had been carried forward with an ISO C standard compiler and the historical function prototype, most applications would have to be modified to cast pointers to the structures actually used to be pointers to **struct qelem** to avoid compilation warnings. By specifying **void** * as the argument type, applications do not need to change (unless they specifically referenced **struct qelem** and depended on it being defined in **<search.h>**).

42536 **RATIONALE**

42537 None.

42538 **FUTURE DIRECTIONS**

42539 None.

42540 **SEE ALSO**

42541 XBD **<search.h>**

42542 **CHANGE HISTORY**

42543 First released in Issue 4, Version 2.

42544 **Issue 5**

42545 Moved from X/OPEN UNIX extension to BASE.

42546 **Issue 6**

42547 The normative text is updated to avoid use of the term ``must'' for application requirements.

NAME

isalnum, isalnum_l — test for an alphanumeric character

SYNOPSIS

```
#include <ctype.h>

int isalnum(int c);
CX int isalnum_l(int c, locale_t locale);
```

DESCRIPTION

CX For *isalnum*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *isalnum*( ) and *isalnum_l*( ) functions shall test whether *c* is a character of class **alpha** or CX **digit** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *c* argument is an **int**, the value of which the application shall ensure is representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *isalnum_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *isalnum*( ) and *isalnum_l*( ) functions shall return non-zero if *c* is an alphanumeric character; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalpha( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<ctype.h>**, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *isalnum_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0274 [302], XSH/TC1-2008/0275 [283], and XSH/TC1-2008/0276 [283] are applied.

NAME

isalpha, isalpha_l — test for an alphabetic character

SYNOPSIS

```
#include <ctype.h>

int isalpha(int c);
CX int isalpha_l(int c, locale_t locale);
```

DESCRIPTION

CX For *isalpha*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *isalpha*( ) and *isalpha_l*( ) functions shall test whether *c* is a character of class **alpha** in the CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *c* argument is an **int**, the value of which the application shall ensure is representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *isalpha_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *isalpha*( ) and *isalpha_l*( ) functions shall return non-zero if *c* is an alphabetic character; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<ctype.h>**, **<locale.h>**, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

42635 **Issue 7**

42636 The *isalpha_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API
42637 Set Part 4.

42638 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0277 [302], XSH/TC1-2008/0278 [283],
42639 and XSH/TC1-2008/0279 [283] are applied.

NAME

isatty — test for a terminal device

SYNOPSIS

```
#include <unistd.h>

int isatty(int fildes);
```

DESCRIPTION

The *isatty*( ) function shall test whether *fildes*, an open file descriptor, is associated with a terminal device.

RETURN VALUE

The *isatty*( ) function shall return 1 if *fildes* is associated with a terminal; otherwise, it shall return 0 and may set *errno* to indicate the error.

ERRORS

The *isatty*( ) function may fail if:

[EBADF] The *fildes* argument is not a valid open file descriptor.

[ENOTTY] The file associated with the *fildes* argument is not a terminal.

EXAMPLES

None.

APPLICATION USAGE

The *isatty*( ) function does not necessarily indicate that a human being is available for interaction via *fildes*. It is quite possible that non-terminal devices are connected to the communications line.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The optional setting of *errno* to indicate an error is added.
- The [EBADF] and [ENOTTY] optional error conditions are added.

Issue 7

SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

NAME

isblank, isblank_l — test for a blank character

SYNOPSIS

```
#include <ctype.h>

int isblank(int c);
CX int isblank_l(int c, locale_t locale);
```

DESCRIPTION

CX For *isblank*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *isblank*( ) and *isblank_l*( ) functions shall test whether *c* is a character of class **blank** in the CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *c* argument is a type **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *isblank_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *isblank*( ) and *isblank_l*( ) functions shall return non-zero if *c* is a <blank>; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

The *isblank_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

42718 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0280 [302], XSH/TC1-2008/0281 [283],
42719 and XSH/TC1-2008/0282 [283] are applied.

NAME

iscntrl, iscntrl_l — test for a control character

SYNOPSIS

```
#include <ctype.h>

int iscntrl(int c);
CX int iscntrl_l(int c, locale_t locale);
```

DESCRIPTION

CX For *iscntrl*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *iscntrl*() and *iscntrl_l*() functions shall test whether *c* is a character of class **cntrl** in the CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *c* argument is a type **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *iscntrl_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *iscntrl*() and *iscntrl_l*() functions shall return non-zero if *c* is a control character; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum(), *isalpha*(), *isblank*(), *isdigit*(), *isgraph*(), *islower*(), *isprint*(), *ispunct*(), *isspace*(), *isupper*(), *isxdigit*(), *setlocale*(), *uselocale*()

XBD Chapter 7 (on page 127), **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *iscntrl_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0283 [302], XSH/TC1-2008/0284 [283], and XSH/TC1-2008/0285 [283] are applied.

NAME

isdigit, isdigit_l — test for a decimal digit

SYNOPSIS

```
#include <ctype.h>

int isdigit(int c);
CX int isdigit_l(int c, locale_t locale);
```

DESCRIPTION

CX For *isdigit*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *isdigit*() and *isdigit_l*() functions shall test whether *c* is a character of class **digit** in the CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *isdigit_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *isdigit*() and *isdigit_l*() functions shall return non-zero if *c* is a decimal digit; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum(), *isalpha*(), *isblank*(), *iscntrl*(), *isgraph*(), *islower*(), *isprint*(), *ispunct*(), *isspace*(), *isupper*(), *isxdigit*()

XBD Chapter 7 (on page 127), **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *isdigit_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0286 [302], XSH/TC1-2008/0287 [283], and XSH/TC1-2008/0288 [283] are applied.

42812 **NAME**

42813 isfinite — test for finite value

42814 **SYNOPSIS**

42815 `#include <math.h>`

42816 `int isfinite(real-floating x);`

42817 **DESCRIPTION**

42818 CX The functionality described on this reference page is aligned with the ISO C standard. Any
42819 conflict between the requirements described here and the ISO C standard is unintentional. This
42820 volume of POSIX.1-2024 defers to the ISO C standard.

42821 The *isfinite*( ) macro shall determine whether its argument has a finite value (zero, subnormal, or
42822 normal, and not infinite or NaN). First, an argument represented in a format wider than its
42823 semantic type is converted to its semantic type. Then determination is based on the type of the
42824 argument.

42825 **RETURN VALUE**

42826 The *isfinite*( ) macro shall return a non-zero value if and only if its argument has a finite value.

42827 **ERRORS**

42828 No errors are defined.

42829 **EXAMPLES**

42830 None.

42831 **APPLICATION USAGE**

42832 None.

42833 **RATIONALE**

42834 None.

42835 **FUTURE DIRECTIONS**

42836 None.

42837 **SEE ALSO**

42838 *fpclassify*( ), *isinf*( ), *isnan*( ), *isnormal*( ), *signbit*( )

42839 XBD **<math.h>**

42840 **CHANGE HISTORY**

42841 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

NAME

isgraph, isgraph_l — test for a visible character

SYNOPSIS

```
#include <ctype.h>

int isgraph(int c);
CX int isgraph_l(int c, locale_t locale);
```

DESCRIPTION

CX For *isgraph*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *isgraph*( ) and *isgraph_l*( ) functions shall test whether *c* is a character of class **graph** in the CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *isgraph_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *isgraph*( ) and *isgraph_l*( ) functions shall return non-zero if *c* is a character with a visible representation; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *isgraph_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0289 [302], XSH/TC1-2008/0290 [283], and XSH/TC1-2008/0291 [283] are applied.

NAME

isgreater, isgreaterequal, isless, islessequal, islessgreater — real-floating relational tests

SYNOPSIS

```
#include <math.h>

int isgreater(real-floating x, real-floating y);
int isgreaterequal(real-floating x, real-floating y);
int isless(real-floating x, real-floating y);
int islessequal(real-floating x, real-floating y);
int islessgreater(real-floating x, real-floating y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *isgreater*( ) macro shall determine whether its first argument is greater than its second argument. The value of *isgreater*(*x*, *y*) shall be equal to (*x*) > (*y*); however, unlike (*x*) > (*y*), *isgreater*(*x*, *y*) shall not raise the invalid floating-point exception when *x* and *y* are unordered.

The *isgreaterequal*( ) macro shall determine whether its first argument is greater than or equal to its second argument. The value of *isgreaterequal*(*x*, *y*) shall be equal to (*x*) ≥ (*y*); however, unlike (*x*) ≥ (*y*), *isgreaterequal*(*x*, *y*) shall not raise the invalid floating-point exception when *x* and *y* are unordered.

The *isless*( ) macro shall determine whether its first argument is less than its second argument. The value of *isless*(*x*, *y*) shall be equal to (*x*) < (*y*); however, unlike (*x*) < (*y*), *isless*(*x*, *y*) shall not raise the invalid floating-point exception when *x* and *y* are unordered.

The *islessequal*( ) macro shall determine whether its first argument is less than or equal to its second argument. The value of *islessequal*(*x*, *y*) shall be equal to (*x*) <= (*y*); however, unlike (*x*) <= (*y*), *islessequal*(*x*, *y*) shall not raise the invalid floating-point exception when *x* and *y* are unordered.

The *islessgreater*( ) macro shall determine whether its first argument is less than or greater than its second argument. The *islessgreater*(*x*, *y*) macro is similar to (*x*) < (*y*) || (*x*) > (*y*); however, *islessgreater*(*x*, *y*) shall not raise the invalid floating-point exception when *x* and *y* are unordered (nor shall it evaluate *x* and *y* twice).

MX Relational operators and their corresponding comparison macros shall produce equivalent result values, even if argument values are represented in wider formats. Thus, comparison macro arguments represented in formats wider than their semantic types shall not be converted to the semantic types, unless the wide evaluation method converts operands of relational operators to their semantic types. The standard wide evaluation methods characterized by FLT_EVAL_METHOD equal to 1 or 2 (see **<float.h>**) do not convert operands of relational operators to their semantic types.

RETURN VALUE

Upon successful completion, the *isgreater*( ) macro shall return the value of (*x*) > (*y*).

Upon successful completion, the *isgreaterequal*( ) macro shall return the value of (*x*) ≥ (*y*).

Upon successful completion, the *isless*( ) macro shall return the value of (*x*) < (*y*).

Upon successful completion, the *islessequal*( ) macro shall return the value of (*x*) <= (*y*).

Upon successful completion, the *islessgreater*( ) macro shall return the value of (*x*) < (*y*) || (*x*) > (*y*).

If *x* or *y* is NaN, these functions shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators may raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for NaNs without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating** indicates that the argument shall be an expression of **real-floating** type.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isunordered( )

XBD **<float.h>**, **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 8

The individual pages for these functions have been merged to form a single page, to reduce duplication.

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

NAME

isinf — test for infinity

SYNOPSIS

```
#include <math.h>

int isinf(real-floating x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *isinf*( ) macro shall determine whether its argument value is an infinity (positive or negative). First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

RETURN VALUE

The *isinf*( ) macro shall return a non-zero value if and only if its argument has an infinite value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fpclassify( ), *isfinite*( ), *isnan*( ), *isnormal*( ), *signbit*( )

XBD **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

42990 **NAME**

42991 isless, islessequal, islessgreater — real-floating relational tests

42992 **SYNOPSIS**

42993
```
#include <math.h>
```

42994
```
int isless(real-floating x, real-floating y);
```
42995
```
int islessequal(real-floating x, real-floating y);
```
42996
```
int islessgreater(real-floating x, real-floating y);
```

42997 **DESCRIPTION**

42998 Refer to *isgreater( )*.

NAME

islower, islower_l — test for a lowercase letter

SYNOPSIS

```
#include <ctype.h>

int islower(int c);
CX int islower_l(int c, locale_t locale);
```

DESCRIPTION

CX For *islower*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *islower*( ) and *islower_l*( ) functions shall test whether *c* is a character of class **lower** in the CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *islower_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *islower*( ) and *islower_l*( ) functions shall return non-zero if *c* is a lowercase letter; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

Testing for a Lowercase Letter

Two examples follow, the first using *islower*( ), the second using multiple concurrent locales and *islower_l*( ).

The examples test whether the value is a lowercase letter, based on the current locale, then use it as part of a key value.

```
/* Example 1 -- using islower() */
#include <ctype.h>
#include <stdlib.h>
#include <locale.h>
...
char *keystr;
int elementlen, len;
unsigned char c;
...
setlocale(LC_ALL, "");
...
len = 0;
while (len < elementlen) {
    c = (unsigned char) (rand() % 256);
...
```



```
            if (islower(c))
                keystr[len++] = c;
            }
        ...
        /* Example 2 -- using islower_l() */
        #include <ctype.h>
        #include <stdlib.h>
        #include <locale.h>
        ...
        char *keystr;
        int elementlen, len;
        unsigned char c;
        ...
        locale_t loc = newlocale (LC_ALL_MASK, "", (locale_t) 0);
        ...
        len = 0;
        while (len < elementlen) {
            c = (unsigned char) (rand() % 256);
        ...
            if (islower_l(c, loc))
                keystr[len++] = c;
            }
        ...
```

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements and an example is added.

Issue 7

The *islower_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

43086 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0292 [302], XSH/TC1-2008/0293 [283],
43087 XSH/TC1-2008/0294 [283], XSH/TC1-2008/0295 [302], and XSH/TC1-2008/0296 [304] are
43088 applied.

NAME

isnan — test for a NaN

SYNOPSIS

```
#include <math.h>

int isnan(real-floating x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *isnan*( ) macro shall determine whether its argument value is a NaN. First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

RETURN VALUE

The *isnan*( ) macro shall return a non-zero value if and only if its argument has a NaN value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fpclassify( ), *isfinite*( ), *isinf*( ), *isnormal*( ), *signbit*( )

XBD **<math.h>**

CHANGE HISTORY

First released in Issue 3.

Issue 5

The DESCRIPTION is updated to indicate the return value when NaN is not supported. This text was previously published in the APPLICATION USAGE section.

Issue 6

Re-written for alignment with the ISO/IEC 9899: 1999 standard.

NAME

isnormal — test for a normal value

SYNOPSIS

```
#include <math.h>

int isnormal(real-floating x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *isnormal*( ) macro shall determine whether its argument value is normal (neither zero, subnormal, infinite, nor NaN). First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

RETURN VALUE

The *isnormal*( ) macro shall return a non-zero value if and only if its argument has a normal value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fpclassify( ), *isfinite*( ), *isinf*( ), *isnan*( ), *signbit*( )

XBD **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

NAME

isprint, isprint_l — test for a printable character

SYNOPSIS

```
#include <ctype.h>

int isprint(int c);
CX int isprint_l(int c, locale_t locale);
```

DESCRIPTION

CX For *isprint*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *isprint*( ) and *isprint_l*( ) functions shall test whether *c* is a character of class **print** in the CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *isprint_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *isprint*( ) and *isprint_l*( ) functions shall return non-zero if *c* is a printable character; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *isprint_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0297 [302], XSH/TC1-2008/0298 [283], and XSH/TC1-2008/0299 [283] are applied.

NAME

ispunct, ispunct_l — test for a punctuation character

SYNOPSIS

```
#include <ctype.h>

int ispunct(int c);
CX int ispunct_l(int c, locale_t locale);
```

DESCRIPTION

CX For *ispunct*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *ispunct*( ) and *ispunct_l*( ) functions shall test whether *c* is a character of class **punct** in the CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *ispunct_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *ispunct*( ) and *ispunct_l*( ) functions shall return non-zero if *c* is a punctuation character; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *ispunct_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0300 [302], XSH/TC1-2008/0301 [283], and XSH/TC1-2008/0302 [283] are applied.

NAME

isspace, isspace_l — test for a white-space character

SYNOPSIS

```
#include <ctype.h>

int isspace(int c);
CX int isspace_l(int c, locale_t locale);
```

DESCRIPTION

CX For *isspace*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *isspace*( ) and *isspace_l*( ) functions shall test whether *c* is a character of class **space** in the CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *isspace_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *isspace*( ) and *isspace_l*( ) functions shall return non-zero if *c* is a white-space character; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *isspace_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0303 [302], XSH/TC1-2008/0304 [283], and XSH/TC1-2008/0305 [283] are applied.

NAME

isunordered — test if arguments are unordered

SYNOPSIS

```
#include <math.h>

int isunordered(real-floating x, real-floating y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *isunordered*( ) macro shall determine whether its arguments are unordered.

RETURN VALUE

Upon successful completion, the *isunordered*( ) macro shall return 1 if its arguments are unordered, and 0 otherwise.

If *x* or *y* is NaN, 1 shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators may raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for NaNs without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating** indicates that the argument shall be an expression of **real-floating** type.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isgreater( )

XBD **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/50 is applied, correcting the RETURN VALUE section when *x* or *y* is NaN.

NAME

isupper, isupper_l — test for an uppercase letter

SYNOPSIS

```
#include <ctype.h>

int isupper(int c);
CX int isupper_l(int c, locale_t locale);
```

DESCRIPTION

CX For *isupper*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *isupper*( ) and *isupper_l*( ) functions shall test whether *c* is a character of class **upper** in the CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *isupper_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *isupper*( ) and *isupper_l*( ) functions shall return non-zero if *c* is an uppercase letter; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *isupper_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0306 [302], XSH/TC1-2008/0307 [283], and XSH/TC1-2008/0308 [283] are applied.

NAME

iswalnum, iswalnum_l — test for an alphanumeric wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswalnum(wint_t wc);
int iswalnum_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswalnum*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *iswalnum*( ) and *iswalnum_l*( ) functions shall test whether *wc* is a wide-character code representing a character of class **alpha** or **digit** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswalnum_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswalnum*( ) and *iswalnum_l*( ) functions shall return non-zero if *wc* is an alphanumeric wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalpha( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<locale.h>**, **<stdio.h>**, **<wctype.h>**

CHANGE HISTORY

First released as a World-wide Portability Interface in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899: 1990/Amendment 1: 1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *iswalnum_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0309 [302], XSH/TC1-2008/0310 [283], and XSH/TC1-2008/0311 [283] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0180 [685] is applied.

NAME

iswalpha, iswalpha_l — test for an alphabetic wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswalpha(wint_t wc);
CX int iswalpha_l(wint_t wc, locale_t locale);
```

DESCRIPTION

CX For *iswalpha*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *iswalpha*( ) and *iswalpha_l*( ) functions shall test whether *wc* is a wide-character code CX representing a character of class **alpha** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *iswalpha_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *iswalpha*( ) and *iswalpha_l*( ) functions shall return non-zero if *wc* is an alphabetic wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<locale.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899: 1990/Amendment 1: 1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *iswalpha_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0312 [302], XSH/TC1-2008/0313 [283], and XSH/TC1-2008/0314 [283] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0181 [685] is applied.

NAME

iswblank, iswblank_l — test for a blank wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswblank(wint_t wc);
int iswblank_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswblank*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *iswblank*( ) and *iswblank_l*( ) functions shall test whether *wc* is a wide-character code representing a character of class **blank** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswblank_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswblank*( ) and *iswblank_l*( ) functions shall return non-zero if *wc* is a blank wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<locale.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

The *iswblank_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0315 [302], XSH/TC1-2008/0316 [283], and XSH/TC1-2008/0317 [283] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0182 [685] is applied.

Issue 8

Austin Group Defect 1770 is applied, changing ``*iswblank*( ) and *iswblank*( ) functions'' to ``*iswblank*( ) and *iswblank_l*( ) functions''.

NAME

iswcntrl, iswcntrl_l — test for a control wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswcntrl(wint_t wc);
int iswcntrl_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswcntrl*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *iswcntrl*( ) and *iswcntrl_l*( ) functions shall test whether *wc* is a wide-character code representing a character of class **cntrl** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswcntrl_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswcntrl*( ) and *iswcntrl_l*( ) functions shall return non-zero if *wc* is a control wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<locale.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899: 1990/Amendment 1: 1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *iswcntrl_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0318 [302], XSH/TC1-2008/0319 [283], and XSH/TC1-2008/0320 [283] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0183 [685] is applied.

NAME

iswctype, iswctype_l — test character for a specified class

SYNOPSIS

```
#include <wctype.h>

int iswctype(wint_t wc, wctype_t charclass);
CX int iswctype_l(wint_t wc, wctype_t charclass,
       locale_t locale);
```

DESCRIPTION

CX For *iswctype*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *iswctype*( ) and *iswctype_l*( ) functions shall determine whether the wide-character code *wc* CX has the character class *charclass*, returning true or false. The *iswctype*( ) and *iswctype_l*( ) functions are defined on WEOF and wide-character codes corresponding to the valid character CX encodings in the current locale, or in the locale represented by *locale*, respectively. If the *wc* argument is not in the domain of the function, the result is undefined. If the value of *charclass* is invalid (that is, not obtained by a call to *wctype*( ) or *charclass* is invalidated by a subsequent call to *setlocale*( ) that has affected category *LC_CTYPE*) the result is unspecified.

CX The behavior is undefined if the *locale* argument to *iswctype_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *iswctype*( ) and *iswctype_l*( ) functions shall return non-zero (true) if and only if *wc* has the CX property described by *charclass*. If *charclass* is (**wctype_t**)0, the *iswctype*( ) and *iswctype_l*( ) functions shall return 0.

ERRORS

No errors are defined.

EXAMPLES

Testing for a Valid Character

```
#include <wctype.h>
...
int yes_or_no;
wint_t wc;
wctype_t valid_class;
...
if ((valid_class=wctype("vowel")) == (wctype_t)0)
    /* Invalid character class. */
yes_or_no=iswctype(wc,valid_class);
```

APPLICATION USAGE

The twelve strings "alnum", "alpha", "blank", "cntrl", "digit", "graph", "lower", "print", "punct", "space", "upper", and "xdigit" are reserved for the standard character classes. In the table below, the functions in the left column are equivalent to the functions in the right column.

```
iswalnum(wc)            iswctype(wc, wctype("alnum"))
iswalnum_l(wc, locale)  iswctype_l(wc, wctype("alnum"), locale)
iswalpha(wc)            iswctype(wc, wctype("alpha"))
```



```
iswalpha_l(wc, locale)  iswctype_l(wc, wctype("alpha"), locale)
iswblank(wc)            iswctype(wc, wctype("blank"))
iswblank_l(wc, locale)  iswctype_l(wc, wctype("blank"), locale)
iswcntrl(wc)            iswctype(wc, wctype("cntrl"))
iswcntrl_l(wc, locale)  iswctype_l(wc, wctype("cntrl"), locale)
iswdigit(wc)            iswctype(wc, wctype("digit"))
iswdigit_l(wc, locale)  iswctype_l(wc, wctype("digit"), locale)
iswgraph(wc)            iswctype(wc, wctype("graph"))
iswgraph_l(wc, locale)  iswctype_l(wc, wctype("graph"), locale)
iswlower(wc)            iswctype(wc, wctype("lower"))
iswlower_l(wc, locale)  iswctype_l(wc, wctype("lower"), locale)
iswprint(wc)            iswctype(wc, wctype("print"))
iswprint_l(wc, locale)  iswctype_l(wc, wctype("print"), locale)
iswpunct(wc)            iswctype(wc, wctype("punct"))
iswpunct_l(wc, locale)  iswctype_l(wc, wctype("punct"), locale)
iswspace(wc)            iswctype(wc, wctype("space"))
iswspace_l(wc, locale)  iswctype_l(wc, wctype("space"), locale)
iswupper(wc)            iswctype(wc, wctype("upper"))
iswupper_l(wc, locale)  iswctype_l(wc, wctype("upper"), locale)
iswxdigit(wc)           iswctype(wc, wctype("xdigit"))
iswxdigit_l(wc, locale) iswctype_l(wc, wctype("xdigit"), locale)
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( ), *wctype*( )

XBD **<locale.h>**, **<wctype.h>**

CHANGE HISTORY

First released as World-wide Portability Interfaces in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The behavior of *charclass* = (**wctype_t**)0 is now described.

An example is added.

A new function, *iswblank*( ), is added to the list in the APPLICATION USAGE.

Issue 7

The *iswctype_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0321 [283] and XSH/TC1-2008/0322 [283] are applied.

43670 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0184 [799] and XSH/TC2-2008/0185
43671 [799] are applied.

43672 **Issue 8**

43673 Austin Group Defect 1302 is applied, aligning the *iswctype*( ) function with the
43674 ISO/IEC 9899:2018 standard.

NAME

iswdigit, iswdigit_l — test for a decimal digit wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswdigit(wint_t wc);
CX int iswdigit_l(wint_t wc, locale_t locale);
```

DESCRIPTION

CX For *iswdigit*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *iswdigit*( ) and *iswdigit_l*( ) functions shall test whether *wc* is a wide-character code CX representing a character of class **digit** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *iswdigit_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *iswdigit*( ) and *iswdigit_l*( ) functions shall return non-zero if *wc* is a decimal digit wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<locale.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899: 1990/Amendment 1: 1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *iswdigit_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0323 [302], XSH/TC1-2008/0324 [283], and XSH/TC1-2008/0325 [283] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0186 [685] is applied.

NAME

iswgraph, iswgraph_l — test for a visible wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswgraph(wint_t wc);
int iswgraph_l(wint_t wc, locale_t locale);
```

DESCRIPTION

CX For *iswgraph*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *iswgraph*( ) and *iswgraph_l*( ) functions shall test whether *wc* is a wide-character code CX representing a character of class **graph** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *iswgraph_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *iswgraph*( ) and *iswgraph_l*( ) functions shall return non-zero if *wc* is a wide-character code with a visible representation; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<locale.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899: 1990/Amendment 1: 1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *iswgraph_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0326 [302], XSH/TC1-2008/0327 [283], and XSH/TC1-2008/0328 [283] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0187 [685] is applied.

NAME

iswlower, iswlower_l — test for a lowercase letter wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswlower(wint_t wc);
int iswlower_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswlower*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *iswlower*( ) and *iswlower_l*( ) functions shall test whether *wc* is a wide-character code representing a character of class **lower** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswlower_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswlower*( ) and *iswlower_l*( ) functions shall return non-zero if *wc* is a lowercase letter wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( ) (on page 2328) 1

XBD Chapter 7 (on page 127), **<locale.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899: 1990/Amendment 1: 1995 (E):

43821 • The SYNOPSIS has been changed to indicate that this function and associated data types
43822 are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

43823 **Issue 6**

43824 The normative text is updated to avoid use of the term ``must'' for application requirements.

43825 **Issue 7**

43826 The *iswlower_l*( ) function is added from The Open Group Technical Standard, 2006, Extended
43827 API Set Part 4.

43828 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0329 [302], XSH/TC1-2008/0330 [283],
43829 and XSH/TC1-2008/0331 [283] are applied.

43830 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0188 [685] is applied.

NAME

iswprint, iswprint_l — test for a printable wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswprint(wint_t wc);
int iswprint_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswprint*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *iswprint*( ) and *iswprint_l*( ) functions shall test whether *wc* is a wide-character code representing a character of class **print** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswprint_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswprint*( ) and *iswprint_l*( ) functions shall return non-zero if *wc* is a printable wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<locale.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899: 1990/Amendment 1: 1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *iswprint_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0332 [302], XSH/TC1-2008/0333 [283], and XSH/TC1-2008/0334 [283] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0189 [685] is applied.

NAME

iswpunct, iswpunct_l — test for a punctuation wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswpunct(wint_t wc);
int iswpunct_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswpunct*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *iswpunct*( ) and *iswpunct_l*( ) functions shall test whether *wc* is a wide-character code representing a character of class **punct** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswpunct_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswpunct*( ) and *iswpunct_l*( ) functions shall return non-zero if *wc* is a punctuation wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<locale.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899: 1990/Amendment 1: 1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *iswpunct_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0335 [302], XSH/TC1-2008/0336 [283], and XSH/TC1-2008/0337 [283] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0190 [685] is applied.

NAME

iswspace, iswspace_l — test for a white-space wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswspace(wint_t wc);
CX int iswspace_l(wint_t wc, locale_t locale);
```

DESCRIPTION

CX For *iswspace*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *iswspace*( ) and *iswspace_l*( ) functions shall test whether *wc* is a wide-character code CX representing a character of class **space** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *iswspace_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *iswspace*( ) and *iswspace_l*( ) functions shall return non-zero if *wc* is a white-space wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<locale.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899: 1990/Amendment 1: 1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *iswspace_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0338 [302], XSH/TC1-2008/0339 [283], and XSH/TC1-2008/0340 [283] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0191 [685] is applied.

NAME

iswupper, iswupper_l — test for an uppercase letter wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswupper(wint_t wc);
CX int iswupper_l(wint_t wc, locale_t locale);
```

DESCRIPTION

CX For *iswupper*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *iswupper*( ) and *iswupper_l*( ) functions shall test whether *wc* is a wide-character code CX representing a character of class **upper** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *iswupper_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *iswupper*( ) and *iswupper_l*( ) functions shall return non-zero if *wc* is an uppercase letter wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<locale.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899: 1990/Amendment 1: 1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *iswupper_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0341 [302], XSH/TC1-2008/0342 [283], and XSH/TC1-2008/0343 [283] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0192 [685] is applied.

NAME

iswxdigit, iswxdigit_l — test for a hexadecimal digit wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswxdigit(wint_t wc);
int iswxdigit_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswxdigit*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *iswxdigit*() and *iswxdigit_l*() functions shall test whether *wc* is a wide-character code representing a character of class **xdigit** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswxdigit_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswxdigit*() and *iswxdigit_l*() functions shall return non-zero if *wc* is a hexadecimal digit wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum(), *iswalpha*(), *iswcntrl*(), *iswctype*(), *iswdigit*(), *iswgraph*(), *iswlower*(), *iswprint*(), *iswpunct*(), *iswspace*(), *iswupper*(), *setlocale*(), *uselocale*()

XBD Chapter 7 (on page 127), **<locale.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *iswxdigit_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0344 [302], XSH/TC1-2008/0345 [283], and XSH/TC1-2008/0346 [283] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0193 [685] is applied.

NAME

isxdigit, isxdigit_l — test for a hexadecimal digit

SYNOPSIS

```
#include <ctype.h>

int isxdigit(int c);
CX int isxdigit_l(int c, locale_t locale);
```

DESCRIPTION

CX For *isxdigit*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *isxdigit*() and *isxdigit_l*() functions shall test whether *c* is a character of class **xdigit** in the CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *isxdigit_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *isxdigit*() and *isxdigit_l*() functions shall return non-zero if *c* is a hexadecimal digit; otherwise, they shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum(), *isalpha*(), *isblank*(), *iscntrl*(), *isdigit*(), *isgraph*(), *islower*(), *isprint*(), *ispunct*(), *isspace*(), *isupper*()

XBD Chapter 7 (on page 127), **<ctype.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *isxdigit_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0347 [302], XSH/TC1-2008/0348 [283], and XSH/TC1-2008/0349 [283] are applied.

NAME

j0, j1, jn — Bessel functions of the first kind

SYNOPSIS

XSI
```
#include <math.h>

double j0(double x);
double j1(double x);
double jn(int n, double x);
```

DESCRIPTION

The *j0*( ), *j1*( ), and *jn*( ) functions shall compute Bessel functions of *x* of the first kind of orders 0, 1, and *n*, respectively.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the relevant Bessel value of *x* of the first kind.

If the *x* argument is finite and too large in magnitude, or the correct result would cause
MXX underflow and is not representable, a range error may occur, and the function shall return 0.0, or (if the IEC 60559 Floating-Point option is not supported) an implementation-defined value no greater in magnitude than DBL_MIN.

MXX If the correct result would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

MXX If x is +Inf, +0 shall be returned.

MXX If *x* is NaN, a NaN shall be returned.

ERRORS

These functions may fail if:

Range Error — The value of *x* was too large in magnitude, or an underflow occurred.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

No other errors shall occur.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( ), *y0*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The may fail [EDOM] error is removed for the case for NaN.

The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling with the ISO/IEC 9899: 1999 standard.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0350 [68] is applied.

Issue 8

Austin Group Defect 714 is applied, changing the behavior of these functions for special cases to be a better match for their mathematical behavior.

NAME

jrand48 — generate a uniformly distributed pseudo-random long signed integer

SYNOPSIS

XSI

```
#include <stdlib.h>

long jrand48(unsigned short xsubi[3]);
```

DESCRIPTION

Refer to *drand48( )*.

NAME

kill — send a signal to a process or a group of processes

SYNOPSIS

CX
```
#include <signal.h>

int kill(pid_t pid, int sig);
```

DESCRIPTION

The *kill*( ) function shall send a signal to a process or a group of processes specified by *pid*. The signal to be sent is specified by *sig* and is either one from the list given in **<signal.h>** or 0. If *sig* is 0 (the null signal), error checking is performed but no signal is actually sent. The null signal can be used to check the validity of *pid*.

For a process to have permission to send a signal to a process designated by *pid*, unless the sending process has appropriate privileges, the real or effective user ID of the sending process shall match the real or saved set-user-ID of the receiving process.

If *pid* is greater than 0, *sig* shall be sent to the process whose process ID is equal to *pid*.

If *pid* is 0, *sig* shall be sent to all processes (excluding an unspecified set of system processes) whose process group ID is equal to the process group ID of the sender, and for which the process has permission to send a signal.

If *pid* is –1, *sig* shall be sent to all processes (excluding an unspecified set of system processes) for which the process has permission to send that signal.

If *pid* is negative, but not –1, *sig* shall be sent to all processes (excluding an unspecified set of system processes) whose process group ID is equal to the absolute value of *pid*, and for which the process has permission to send a signal.

If the value of *pid* causes *sig* to be generated for the sending process, and if *sig* is not blocked for the calling thread and if no other thread has *sig* unblocked or is waiting in a *sigwait*( ) function for *sig*, either *sig* or at least one pending unblocked signal shall be delivered to the sending thread before *kill*( ) returns.

The user ID tests described above shall not be applied when sending SIGCONT to a process that is a member of the same session as the sending process.

An implementation that provides extended security controls may impose further implementation-defined restrictions on the sending of signals, including the null signal. In particular, the system may deny the existence of some or all of the processes specified by *pid*.

The *kill*( ) function is successful if the process has permission to send *sig* to any of the processes specified by *pid*. If *kill*( ) fails, no signal shall be sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *kill*( ) function shall fail if:

[EINVAL] The value of the *sig* argument is an invalid or unsupported signal number.

[EPERM] The process does not have permission to send the signal to any receiving process.

[ESRCH] No process or process group can be found corresponding to that specified by *pid*.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The semantics for permission checking for *kill*( ) differed between System V and most other implementations, such as Version 7 or 4.3 BSD. The semantics chosen for this volume of POSIX.1-2024 agree with System V. Specifically, a set-user-ID process cannot protect itself against signals (or at least not against SIGKILL) unless it changes its real user ID. This choice allows the user who starts an application to send it signals even if it changes its effective user ID. The other semantics give more power to an application that wants to protect itself from the user who ran it.

Some implementations provide semantic extensions to the *kill*( ) function when the absolute value of *pid* is greater than some maximum, or otherwise special, value. Negative values are a flag to *kill*( ). Since most implementations return [ESRCH] in this case, this behavior is not included in this volume of POSIX.1-2024, although a conforming implementation could provide such an extension.

The unspecified processes to which a signal cannot be sent may include the scheduler or *init*.

There was initially strong sentiment to specify that, if *pid* specifies that a signal be sent to the calling process and that signal is not blocked, that signal would be delivered before *kill*( ) returns. This would permit a process to call *kill*( ) and be guaranteed that the call never return. However, historical implementations that provide only the *signal*( ) function make only the weaker guarantee in this volume of POSIX.1-2024, because they only deliver one signal each time a process enters the kernel. Modifications to such implementations to support the *sigaction*( ) function generally require entry to the kernel following return from a signal-catching function, in order to restore the signal mask. Such modifications have the effect of satisfying the stronger requirement, at least when *sigaction*( ) is used, but not necessarily when *signal*( ) is used. The standard developers considered making the stronger requirement except when *signal*( ) is used, but felt this would be unnecessarily complex. Implementors are encouraged to meet the stronger requirement whenever possible. In practice, the weaker requirement is the same, except in the rare case when two signals arrive during a very short window. This reasoning also applies to a similar requirement for *sigprocmask*( ).

In 4.2 BSD, the SIGCONT signal can be sent to any descendant process regardless of user-ID security checks. This allows a job control shell to continue a job even if processes in the job have altered their user IDs (as in the *su* command). In keeping with the addition of the concept of sessions, similar functionality is provided by allowing the SIGCONT signal to be sent to any process in the same session regardless of user ID security checks. This is less restrictive than BSD in the sense that ancestor processes (in the same session) can now be the recipient. It is more restrictive than BSD in the sense that descendant processes that form new sessions are now subject to the user ID checks. A similar relaxation of security is not necessary for the other job control signals since those signals are typically sent by the terminal driver in recognition of special characters being typed; the terminal driver bypasses all security checks.

In secure implementations, a process may be restricted from sending a signal to a process having a different security label. In order to prevent the existence or nonexistence of a process from being used as a covert channel, such processes should appear nonexistent to the sender; that is, [ESRCH] should be returned, rather than [EPERM], if *pid* refers only to such processes.

Historical implementations varied on the result of a *kill*( ) with *pid* indicating a zombie process. Some indicated success on such a call (subject to permission checking), while others gave an error of [ESRCH]. Since the definition of process lifetime in this volume of POSIX.1-2024 covers zombie processes, the [ESRCH] error as described is inappropriate in this case and implementations that give this error do not conform. This means that an application cannot have a parent process check for termination of a particular child by sending it the null signal with *kill*( ), but must instead use *waitpid*( ) or *waitid*( ).

There is some belief that the name *kill*( ) is misleading, since the function is not always intended to cause process termination. However, the name is common to all historical implementations, and any change would be in conflict with the goal of minimal changes to existing application code.

FUTURE DIRECTIONS

None.

SEE ALSO

getpid( ), *raise*( ), *setsid*( ), *sig2str*( ), *sigaction*( ), *sigqueue*( ), *wait*( )

XBD **<signal.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the second paragraph is reworded to indicate that the saved set-user-ID of the calling process is checked in place of its effective user ID. This is a FIPS requirement.
- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The behavior when *pid* is –1 is now specified. It was previously explicitly unspecified in the POSIX.1-1988 standard.

The normative text is updated to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/51 is applied, correcting the RATIONALE section.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0194 [765] is applied.

Issue 8

Austin Group Defect 1138 is applied, adding *sig2str*( ) to the SEE ALSO section.

NAME

kill_dependency — terminate a dependency chain

SYNOPSIS

```
#include <stdatomic.h>

type kill_dependency(type *y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

Implementations that define the macro __STDC_NO_ATOMICS__ need not provide the **<stdatomic.h>** header nor support the *kill_dependency*( ) macro.

The *kill_dependency*( ) macro shall terminate a dependency chain (see XBD Section 4.15.1, on page 100). The argument shall not carry a dependency to the return value.

RETURN VALUE

The *kill_dependency*( ) macro shall return the value of *y*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD Section 4.15.1, **<stdatomic.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

NAME

killpg — send a signal to a process group

SYNOPSIS

XSI
```
#include <signal.h>

int killpg(pid_t pgrp, int sig);
```

DESCRIPTION

The *killpg*( ) function shall send the signal specified by *sig* to the process group specified by *pgrp*.

If *pgrp* is greater than 1, *killpg*(*pgrp*, *sig*) shall be equivalent to *kill*(–*pgrp*, *sig*). If *pgrp* is less than or equal to 1, the behavior of *killpg*( ) is undefined.

RETURN VALUE

Refer to *kill*( ).

ERRORS

Refer to *kill*( ).

EXAMPLES

Sending a Signal to All Other Members of a Process Group

The following example shows how the calling process could send a signal to all other members of its process group. To prevent itself from receiving the signal it first makes itself immune to the signal by ignoring it.

```
#include <signal.h>
#include <unistd.h>
...
    if (signal(SIGUSR1, SIG_IGN) == SIG_ERR)
        /* Handle error */;

    if (killpg(getpgrp(), SIGUSR1) == -1)
        /* Handle error */;"
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getpgid( ), *getpid*( ), *kill*( ), *raise*( )

XBD **<signal.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/52 is applied, adding the example to the EXAMPLES section.

NAME

l64a — convert a 32-bit integer to a radix-64 ASCII string

SYNOPSIS

XSI

```
#include <stdlib.h>

char *l64a(long value);
```

DESCRIPTION

Refer to *a64l( )*.

NAME

labs, llabs — return a long integer absolute value

SYNOPSIS

```
#include <stdlib.h>

long labs(long i);
long long llabs(long long i);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *labs*( ) function shall compute the absolute value of the **long** integer operand *i*. The *llabs*( ) function shall compute the absolute value of the **long long** integer operand *i*. If the result cannot be represented, the behavior is undefined.

RETURN VALUE

The *labs*( ) function shall return the absolute value of the **long** integer operand.

The *llabs*( ) function shall return the absolute value of the **long long** integer operand.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Since POSIX.1 requires a two's complement representation of **long** and **long long**, the absolute value of the negative integers with the largest magnitude {LONG_MIN} and {LLONG_MIN} are not representable, thus *labs*(LONG_MIN) and *llabs*(LLONG_MIN) are undefined.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

abs( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ISO C standard.

Issue 6

The *llabs*( ) function is added for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

SD5-XSH-ERN-152 is applied, correcting the RETURN VALUE section.

Issue 8

Austin Group Defect 1108 is applied, changing the APPLICATION USAGE section.

NAME

lchown — change the owner and group of a symbolic link

SYNOPSIS

```
#include <unistd.h>

int lchown(const char *path, uid_t owner, gid_t group);
```

DESCRIPTION

The *lchown*( ) function shall be equivalent to *chown*( ), except in the case where the named file is a symbolic link. In this case, *lchown*( ) shall change the ownership of the symbolic link file itself, while *chown*( ) changes the ownership of the file or directory to which the symbolic link refers.

RETURN VALUE

Upon successful completion, *lchown*( ) shall return 0. Otherwise, it shall return –1 and set *errno* to indicate an error.

ERRORS

The *lchown*( ) function shall fail if:

[EACCES] Search permission is denied on a component of the path prefix of *path*.

[EINVAL] The owner or group ID is not a value supported by the implementation.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of a component of a pathname is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

[EPERM] The effective user ID does not match the owner of the file and the process does not have appropriate privileges.

[EROFS] The file resides on a read-only file system.

The *lchown*( ) function may fail if:

[EIO] An I/O error occurred while reading or writing to the file system.

[EINTR] A signal was caught during execution of the function.

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

EXAMPLES

Changing the Current Owner of a File

The following example shows how to change the ownership of the symbolic link named **/modules/pass1** to the user ID associated with ``jones'' and the group ID associated with ``cnd''.

The numeric value for the user ID is obtained by using the *getpwnam*( ) function. The numeric value for the group ID is obtained by using the *getgrnam*( ) function.

```
#include <sys/types.h>
#include <unistd.h>
#include <pwd.h>
#include <grp.h>

struct passwd *pwd;
struct group  *grp;
char          *path = "/modules/pass1";
...
pwd = getpwnam("jones");
grp = getgrnam("cnd");
lchown(path, pwd->pw_uid, grp->gr_gid);
```

APPLICATION USAGE

On implementations which support symbolic links as directory entries rather than files, *lchown*( ) may fail.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

chown( ), *symlink*( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

The Open Group Base Resolution bwg2001-013 is applied, adding wording to the APPLICATION USAGE.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

The *lchown*( ) function is moved from the XSI option to the Base.

The [EOPNOTSUPP] error is removed.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

44527 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0351 [324] is applied.

NAME

lcong48 — seed a uniformly distributed pseudo-random signed long integer generator

SYNOPSIS

XSI

```
#include <stdlib.h>

void lcong48(unsigned short param[7]);
```

DESCRIPTION

Refer to *drand48( )*.

NAME

ldexp, ldexpf, ldexpl — load exponent of a floating-point number

SYNOPSIS

```
#include <math.h>

double ldexp(double x, int exp);
float ldexpf(float x, int exp);
long double ldexpl(long double x, int exp);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the quantity $x * 2^{exp}$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return *x* multiplied by 2, raised to the power *exp*.

If these functions would cause overflow, a range error shall occur and *ldexp*( ), *ldexpf*( ), and *ldexpl*( ) shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (according to the sign of *x*), respectively.

MXX If the correct value would cause underflow, and is not representable, a range error may occur, and *ldexp*( ), *ldexpf*( ), and *ldexpl*( ) shall return 0.0, or (if IEC 60559 Floating-Point is not supported) an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

If *exp* is 0, *x* shall be returned.

MXX If the correct value would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

ERRORS

These functions shall fail if:

Range Error The result overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

Range Error The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow

floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *frexp*( ), *isnan*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *ldexpf*( ) and *ldexpl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0352 [68] and XSH/TC1-2008/0353 [68] are applied.

NAME

ldiv, lldiv — compute quotient and remainder of a long division

SYNOPSIS

```
#include <stdlib.h>

ldiv_t ldiv(long numer, long denom);
lldiv_t lldiv(long long numer, long long denom);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the quotient and remainder of the division of the numerator *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the **long** integer (for the *ldiv*( ) function) or **long long** integer (for the *lldiv*( ) function) of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is undefined; otherwise, *quot* * *denom*+*rem* shall equal *numer*.

RETURN VALUE

The *ldiv*( ) function shall return a structure of type **ldiv_t**, comprising both the quotient and the remainder. The structure shall include the following members, in any order:

```
long   quot;    /* Quotient */
long   rem;     /* Remainder */
```

The *lldiv*( ) function shall return a structure of type **lldiv_t**, comprising both the quotient and the remainder. The structure shall include the following members, in any order:

```
long long   quot;    /* Quotient */
long long   rem;     /* Remainder */
```

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

div( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ISO C standard.

Issue 6

The *lldiv*( ) function is added for alignment with the ISO/IEC 9899: 1999 standard.

NAME

le16toh, le32toh, le64toh — convert values between host and specified byte order

SYNOPSIS

```
#include <endian.h>

uint16_t le16toh(uint16_t little_endian_16bits);
uint32_t le32toh(uint32_t little_endian_32bits);
uint64_t le64toh(uint64_t little_endian_64bits);
```

DESCRIPTION

Refer to *be16toh( )*.

NAME

lfind — find entry in a linear search table

SYNOPSIS

XSI

```
#include <search.h>

void *lfind(const void *key, const void *base, size_t *nelp,
    size_t width, int (*compar)(const void *, const void *));
```

DESCRIPTION

Refer to *lsearch( )*.

NAME

lgamma, lgammaf, lgammal, signgam — log gamma function

SYNOPSIS

```
#include <math.h>

double lgamma(double x);
float lgammaf(float x);
long double lgammal(long double x);
extern int signgam;
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute $\log_e|\Gamma(x)|$ where $\Gamma(x)$ is defined as $\int_0^\infty e^{-t}t^{x-1}dt$. The argument x need not be a non-positive integer ($\Gamma(x)$ is defined over the reals, except the non-positive integers).

XSI The sign of $\Gamma(x)$ shall be returned in the external integer *signgam*. If x is NaN, –Inf, or a negative integer, the value of *signgam* is unspecified.

If concurrent calls are made to these functions, the value of *signgam* is indeterminate.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the logarithmic gamma of x.

If x is a non-positive integer, a pole error shall occur and *lgamma*( ), *lgammaf*( ), and *lgammal*( ) shall return +HUGE_VAL, +HUGE_VALF, and +HUGE_VALL, respectively.

If the correct value would cause overflow, a range error shall occur and *lgamma*( ), *lgammaf*( ), and *lgammal*( ) shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (having the same sign as the correct value), respectively.

MX If x is NaN, a NaN shall be returned.

If x is 1 or 2, +0 shall be returned.

If x is ±Inf, +Inf shall be returned.

ERRORS

These functions shall fail if:

Pole Error — The x argument is a negative integer or zero.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

Range Error — The result overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

If the value of *signgam* will be obtained after a call to *lgamma*( ), *lgammaf*( ), or *lgammal*( ), in order to ensure that the value will not be altered by another call in a different thread, applications should either restrict calls to these functions to be from a single thread or use a lock such as a mutex or spin lock to protect a critical section starting before the function call and ending after the value of *signgam* has been obtained.

RATIONALE

Earlier versions of this standard did not require *lgamma*( ), *lgammaf*( ), and *lgammal*( ) to be thread-safe because *signgam* was a global variable. They are now required to be thread-safe to align with the ISO C standard (which, since the introduction of threads in 2011, requires that they avoid data races), with the exception that they need not avoid data races when storing a value in the *signgam* variable. Since *signgam* is not specified by the ISO C standard, this exception is not a conflict with that standard.

FUTURE DIRECTIONS

None.

SEE ALSO

exp( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 3.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *lgamma*( ) function is no longer marked as an extension.

The *lgammaf*( ) and *lgammal*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

Functionality relating to the XSI option is marked.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

The DESCRIPTION is clarified regarding the value of *signgam* when *x* is Nan, –Inf, or a negative integer.

Issue 8

Austin Group Defect 1002 is applied, reinstating the requirement for the sign of $\Gamma(x)$ to be returned in *signgam*, which had been accidentally removed in Issue 7.

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018 standard.

NAME

link, linkat — hard link one file to another file

SYNOPSIS

```
#include <unistd.h>

int link(const char *path1, const char *path2);
```

OH
```
#include <fcntl.h>

int linkat(int fd1, const char *path1, int fd2,
    const char *path2, int flag);
```

DESCRIPTION

The *link*( ) function shall create a new hard link (directory entry) for the existing file, *path1*.

The *path1* argument points to a pathname naming an existing file. The *path2* argument points to a pathname naming the new directory entry to be created. The *link*( ) function shall atomically create a new hard link for the existing file and the link count of the file shall be incremented by one.

If *path1* names a directory, *link*( ) shall fail unless the process has appropriate privileges and the implementation supports using *link*( ) on directories.

If *path1* names a symbolic link, it is implementation-defined whether *link*( ) follows the symbolic link, or creates a new hard link to the symbolic link itself.

Upon successful completion, *link*( ) shall mark for update the last file status change timestamp of the file. Also, the last data modification and last file status change timestamps of the directory that contains the new entry shall be marked for update.

If *link*( ) fails, no link shall be created and the link count of the file shall remain unchanged.

The implementation may require that the calling process has permission to access the existing file.

The *linkat*( ) function shall be equivalent to the *link*( ) function except that symbolic links shall be handled as specified by the value of *flag* (see below) and except in the case where either *path1* or *path2* or both are relative paths. In this case a relative path *path1* is interpreted relative to the directory associated with the file descriptor *fd1* instead of the current working directory and similarly for *path2* and the file descriptor *fd2*. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in **<fcntl.h>**:

AT_SYMLINK_FOLLOW
: If *path1* names a symbolic link, a new hard link for the target of the symbolic link is created.

If *linkat*( ) is passed the special value AT_FDCWD in the *fd1* or *fd2* parameter, the current working directory shall be used for the respective *path* argument. If both *fd1* and *fd2* have value AT_FDCWD, the behavior shall be identical to a call to *link*( ), except that symbolic links shall be handled as specified by the value of *flag*.

If the AT_SYMLINK_FOLLOW flag is clear in the *flag* argument and the *path1* argument names a symbolic link, a new hard link is created for the symbolic link *path1* and not its target.

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return –1 and set *errno* to indicate the error.

ERRORS

These functions shall fail if:

[EACCES] A component of either path prefix denies search permission, or the requested link requires writing in a directory that denies write permission, or the calling process does not have permission to access the existing file and this is required by the implementation.

[EEXIST] The *path2* argument resolves to an existing directory entry or refers to a symbolic link.

[EILSEQ] The last pathname component of *path2* is not a portable filename, and cannot be created in the target directory.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path1* or *path2* argument.

[EMLINK] The number of hard links to the file named by *path1* would exceed {LINK_MAX}.

[ENAMETOOLONG]
The length of a component of a pathname is longer than {NAME_MAX}.

[ENOENT] A component of either path prefix does not exist; the file named by *path1* does not exist; or *path1* or *path2* points to an empty string.

[ENOENT] or [ENOTDIR]
The *path1* argument names an existing non-directory file, and the *path2* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters. If *path2* without the trailing <slash> characters would name an existing file, an [ENOENT] error shall not occur.

[ENOSPC] The directory to contain the link cannot be extended.

[ENOTDIR] A component of either path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path1* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory, or the *path1* argument names an existing non-directory file and the *path2* argument names a nonexistent file, contains at least one non-<slash> character, and ends with one or more trailing <slash> characters.

[EPERM] The file named by *path1* is a directory and either the calling process does not have appropriate privileges or the implementation prohibits using *link*( ) on directories.

[EROFS] The requested link requires writing in a directory on a read-only file system.

[EXDEV] The file named by *path1* and the directory in which the directory entry named by *path2* is to be created are on different file systems and the implementation does not support hard links between file systems.

The *linkat*( ) function shall fail if:

[EACCES] The access mode of the open file description associated with *fd1* or *fd2* is not O_SEARCH and the permissions of the directory underlying *fd1* or *fd2*, respectively, do not permit directory searches.

[EBADF] The *path1* or *path2* argument does not specify an absolute path and the *fd1* or *fd2* argument, respectively, is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

[ENOTDIR] The *path1* or *path2* argument is not an absolute path and *fd1* or *fd2*, respectively, is a file descriptor associated with a non-directory file.

These functions may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path1* or *path2* argument.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The *linkat*( ) function may fail if:

[EINVAL] The value of the *flag* argument is not valid.

EXAMPLES

Creating a Hard Link to a File

The following example shows how to create an additional hard link to a file named **/home/cnd/mod1** by creating a new directory entry named **/modules/pass1**.

```
#include <unistd.h>
char *path1 = "/home/cnd/mod1";
char *path2 = "/modules/pass1";
int   status;
...
status = link (path1, path2);
```

Creating a Hard Link to a File Within a Program

In the following program example, the *link*( ) function hard links the **/etc/passwd** file (defined as **PASSWDFILE**) to a file named **/etc/opasswd** (defined as **SAVEFILE**), which is used to save the current password file. Then, after removing the current password file (defined as **PASSWDFILE**), the new password file is saved as the current password file using the *link*( ) function again.

```
#include <unistd.h>

#define LOCKFILE "/etc/ptmp"
#define PASSWDFILE "/etc/passwd"
#define SAVEFILE "/etc/opasswd"
...
/* Save current password file */
link (PASSWDFILE, SAVEFILE);

/* Remove current password file. */
unlink (PASSWDFILE);
```



```
        /* Save new password file as current password file. */
        link (LOCKFILE,PASSWDFILE);
```

APPLICATION USAGE

Some implementations do allow hard links between file systems.

If *path1* refers to a symbolic link, application developers should use *linkat*( ) with appropriate flags to select whether or not the symbolic link should be resolved.

RATIONALE

Creating additional hard links to a directory is restricted to the superuser in most historical implementations because this capability may produce loops in the file hierarchy or otherwise corrupt the file system. This volume of POSIX.1-2024 continues that philosophy by prohibiting *link*( ) and *unlink*( ) from doing this. Other functions could do it if the implementor designed such an extension.

Some historical implementations allow hard linking of files on different file systems. Wording was added to explicitly allow this optional behavior.

The exception for cross-file system hard links is intended to apply only to links that are programmatically indistinguishable from traditional hard links.

The purpose of the *linkat*( ) function is to link files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *link*( ), resulting in unspecified behavior. By opening a file descriptor for the directory of both the existing file and the target location and using the *linkat*( ) function it can be guaranteed that the both filenames are in the desired directories.

Earlier versions of this standard specified only the *link*( ) function, and required it to behave like *linkat*( ) with the AT_SYMLINK_FOLLOW flag. However, historical practice from SVR4 and Linux kernels had *link*( ) behaving like *linkat*( ) with no flags, and many systems that attempted to provide a conforming *link*( ) function did so in a way that was rarely used, and when it was used did not conform to the standard (e.g., by not being atomic, or by dereferencing the symbolic link incorrectly). Since applications could not rely on *link*( ) following symbolic links in practice, the *linkat*( ) function was added taking a flag to specify the desired behavior for the application.

Implementations are encouraged to have *link*( ) and *linkat*( ) report an [EILSEQ] error if the file named by *path2* did not previously exist, and the last component of that pathname contains any bytes that have the encoded value of a <newline> character.

FUTURE DIRECTIONS

None.

SEE ALSO

rename( ), *symlink*( ), *unlink*( )

XBD **<fcntl.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [ELOOP] mandatory error condition is added.

- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- An explanation is added of the action when *path2* refers to a symbolic link.
- The [ELOOP] optional error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

SD5-XSH-ERN-93 is applied, adding RATIONALE.

The *linkat*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Functionality relating to XSI STREAMS is marked obsolescent.

Changes are made related to support for finegrained timestamps.

The [EOPNOTSUPP] error is removed.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0354 [326], XSH/TC1-2008/0355 [461], XSH/TC1-2008/0356 [326], XSH/TC1-2008/0357 [324], XSH/TC1-2008/0358 [147,429], XSH/TC1-2008/0359 [277], XSH/TC1-2008/0360 [278], and XSH/TC1-2008/0361 [278] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0195 [873], XSH/TC2-2008/0196 [591], XSH/TC2-2008/0197 [817], XSH/TC2-2008/0198 [822], and XSH/TC2-2008/0199 [817] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 293 is applied, adding the [EILSEQ] error.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1380 is applied, changing text using the term ``link'' in line with its updated definition and removing a paragraph from the RATIONALE section.

NAME

lio_listio — list directed I/O

SYNOPSIS

```
#include <aio.h>

int lio_listio(int mode, struct aiocb *restrict const list[restrict],
    int nent, struct sigevent *restrict sig);
```

DESCRIPTION

The *lio_listio*( ) function shall initiate a list of I/O requests with a single function call.

The *mode* argument takes one of the values LIO_WAIT or LIO_NOWAIT declared in **<aio.h>** and determines whether the function returns when the I/O operations have been completed, or as soon as the operations have been queued. If the *mode* argument is LIO_WAIT, the function shall wait until all I/O is complete and the *sig* argument shall be ignored.

If the *mode* argument is LIO_NOWAIT, the function shall return immediately, and asynchronous notification shall occur, according to the *sig* argument, when all the I/O operations complete. If *sig* is NULL, then no asynchronous notification shall occur. If *sig* is not NULL, asynchronous notification occurs as specified in Section 2.4.1 (on page 513) when all the requests in *list* have completed.

The I/O requests enumerated by *list* are submitted in an unspecified order.

The *list* argument is an array of pointers to **aiocb** structures. The array contains *nent* elements. The array may contain NULL elements, which shall be ignored.

If the buffer pointed to by *list* or the **aiocb** structures pointed to by the elements of the array *list* become illegal addresses before all asynchronous I/O completed and, if necessary, the notification is sent, then the behavior is undefined. If the buffers pointed to by the *aio_buf* member of the **aiocb** structure pointed to by the elements of the array *list* become illegal addresses prior to the asynchronous I/O associated with that **aiocb** structure being completed, the behavior is undefined.

The *aio_lio_opcode* field of each **aiocb** structure specifies the operation to be performed. The supported operations are LIO_READ, LIO_WRITE, and LIO_NOP; these symbols are defined in **<aio.h>**. The LIO_NOP operation causes the list entry to be ignored. If the *aio_lio_opcode* element is equal to LIO_READ, then an I/O operation is submitted as if by a call to *aio_read*( ) with the *aiocbp* equal to the address of the **aiocb** structure. If the *aio_lio_opcode* element is equal to LIO_WRITE, then an I/O operation is submitted as if by a call to *aio_write*( ) with the *aiocbp* equal to the address of the **aiocb** structure.

The *aio_fildes* member specifies the file descriptor on which the operation is to be performed.

The *aio_buf* member specifies the address of the buffer to or from which the data is transferred.

The *aio_nbytes* member specifies the number of bytes of data to be transferred.

The members of the **aiocb** structure further describe the I/O operation to be performed, in a manner identical to that of the corresponding **aiocb** structure when used by the *aio_read*( ) and *aio_write*( ) functions.

The *nent* argument specifies how many elements are members of the list; that is, the length of the array.

The behavior of this function is altered according to the definitions of synchronized I/O data integrity completion and synchronized I/O file integrity completion if synchronized I/O is enabled on the file associated with *aio_fildes*.

For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *aiocbp*−>*aio_fildes*.

If *sig*−>*sigev_notify* is SIGEV_THREAD and *sig*−>*sigev_notify_attributes* is a non-null pointer and the block pointed to by this pointer becomes an illegal address prior to all asynchronous I/O being completed, then the behavior is undefined.

RETURN VALUE

If the *mode* argument has the value LIO_NOWAIT, the *lio_listio*( ) function shall return the value zero if the I/O operations are successfully queued; otherwise, the function shall return the value −1 and set *errno* to indicate the error.

If the *mode* argument has the value LIO_WAIT, the *lio_listio*( ) function shall return the value zero when all the indicated I/O has completed successfully. Otherwise, *lio_listio*( ) shall return a value of −1 and set *errno* to indicate the error.

In either case, the return value only indicates the success or failure of the *lio_listio*( ) call itself, not the status of the individual I/O requests. In some cases one or more of the I/O requests contained in the list may fail. Failure of an individual request does not prevent completion of any other individual request. To determine the outcome of each I/O request, the application shall examine the error status associated with each **aiocb** control block. The error statuses so returned are identical to those returned as the result of an *aio_read*( ) or *aio_write*( ) function.

ERRORS

The *lio_listio*( ) function shall fail if:

[EAGAIN] The resources necessary to queue all the I/O requests were not available. The application may check the error status for each **aiocb** to determine the individual request(s) that failed.

[EAGAIN] The number of entries indicated by *nent* would cause the system-wide limit {AIO_MAX} to be exceeded.

[EINVAL] The *mode* argument is not a proper value, or the value of *nent* was greater than {AIO_LISTIO_MAX}.

[EINTR] A signal was delivered while waiting for all I/O requests to complete during an LIO_WAIT operation. Note that, since each I/O operation invoked by *lio_listio*( ) may possibly provoke a signal when it completes, this error return may be caused by the completion of one (or more) of the very I/O operations being awaited. Outstanding I/O requests are not canceled, and the application shall examine each list element to determine whether the request was initiated, canceled, or completed.

[EIO] One or more of the individual I/O operations failed. The application may check the error status for each **aiocb** structure to determine the individual request(s) that failed.

If the *lio_listio*( ) function succeeds or fails with errors of [EAGAIN], [EINTR], or [EIO], then some of the I/O specified by the list may have been initiated. If the *lio_listio*( ) function fails with an error code other than [EAGAIN], [EINTR], or [EIO], no operations from the list shall have been initiated. The I/O operation indicated by each list element can encounter errors specific to the individual read or write function being performed. In this event, the error status for each **aiocb** control block contains the associated error code. The error codes that can be set are the same as would be set if the I/O operation had been initiated by an *aio_read*( ) or *aio_write*( ) function, with the following additional error codes possible:

45039 [EAGAIN] The requested I/O operation was not queued due to resource limitations.

45040 [EINPROGRESS] The requested I/O is in progress.

45041 **EXAMPLES**

45042 None.

45043 **APPLICATION USAGE**

45044 None.

45045 **RATIONALE**

45046 Although it may appear that there are inconsistencies in the specified circumstances for error
45047 codes, the [EIO] error condition applies when any circumstance relating to an individual
45048 operation makes that operation fail. This might be due to a badly formulated request (for
45049 example, the *aio_lio_opcode* field is invalid, and *aio_error*( ) returns [EINVAL]) or might arise from
45050 application behavior (for example, the file descriptor is closed before the operation is initiated,
45051 and *aio_error*( ) returns [EBADF]).

45052 The limitation on the set of error codes returned when operations from the list shall have been
45053 initiated enables applications to know when operations have been started and whether
45054 *aio_error*( ) is valid for a specific operation.

45055 **FUTURE DIRECTIONS**

45056 None.

45057 **SEE ALSO**

45058 *aio_read*( ), *aio_write*( ), *aio_error*( ), *aio_return*( ), *aio_cancel*( ), *close*( ), *exec*, *exit*( ), *fork*( ), *lseek*( ),
45059 *read*( )

45060 XBD **<aio.h>**

45061 **CHANGE HISTORY**

45062 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

45063 Large File Summit extensions are added.

45064 **Issue 6**

45065 The [ENOSYS] error condition has been removed as stubs need not be provided if an
45066 implementation does not support the Asynchronous Input and Output option.

45067 The *lio_listio*( ) function is marked as part of the Asynchronous Input and Output option.

45068 The following new requirements on POSIX implementations derive from alignment with the
45069 Single UNIX Specification:

45070 • In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs
45071 past the offset maximum established in the open file description associated with
45072 *aiocbp->aio_fildes*. This change is to support large files.

45073 • The [EBIG] and [EOVERFLOW] error conditions are defined. This change is to support
45074 large files.

45075 The normative text is updated to avoid use of the term ``must'' for application requirements.

45076 The **restrict** keyword is added to the *lio_listio*( ) prototype for alignment with the
45077 ISO/IEC 9899: 1999 standard.

Issue 6

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/53 is applied, adding new text for symmetry with the *aio_read*( ) and *aio_write*( ) functions to the DESCRIPTION.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/54 is applied, adding text to the DESCRIPTION making it explicit that the user is required to keep the structure pointed to by *sig*->*sigev_notify_attributes* valid until the last asynchronous operation finished and the notification has been sent.

Issue 7

The *lio_listio*( ) function is moved from the Asynchronous Input and Output option to the Base.

Issue 8

Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

45089 **NAME**

45090 listen — listen for socket connections and limit the queue of incoming connections

45091 **SYNOPSIS**

45092 ```
#include <sys/socket.h>
```

45093 ```
int listen(int socket, int backlog);
```

45094 **DESCRIPTION**

45095 The *listen*( ) function shall mark a connection-mode socket, specified by the *socket* argument, as
45096 accepting connections.

45097 The *backlog* argument provides a hint to the implementation which the implementation shall use
45098 to limit the number of outstanding connections in the socket's listen queue. Implementations
45099 may impose a limit on *backlog* and silently reduce the specified value. Normally, a larger *backlog*
45100 argument value shall result in a larger or equal length of the listen queue. Implementations shall
45101 support values of *backlog* up to SOMAXCONN, defined in **<sys/socket.h>**.

45102 The implementation may include incomplete connections in its listen queue. The limits on the
45103 number of incomplete connections and completed connections queued may be different.

45104 The implementation may have an upper limit on the length of the listen queue—either global or
45105 per accepting socket. If *backlog* exceeds this limit, the length of the listen queue is set to the limit.

45106 If *listen*( ) is called with a *backlog* argument value that is less than 0, the function behaves as if it
45107 had been called with a *backlog* argument value of 0.

45108 A *backlog* argument of 0 may allow the socket to accept connections, in which case the length of
45109 the listen queue may be set to an implementation-defined minimum value.

45110 The socket in use may require the process to have appropriate privileges to use the *listen*( )
45111 function.

45112 **RETURN VALUE**

45113 Upon successful completions, *listen*( ) shall return 0; otherwise, –1 shall be returned and *errno* set
45114 to indicate the error.

45115 **ERRORS**

45116 The *listen*( ) function shall fail if:

45117 [EBADF] The *socket* argument is not a valid file descriptor.

45118 [EDESTADDRREQ]
45119 The socket is not bound to a local address, and the protocol does not support
45120 listening on an unbound socket.

45121 [EINVAL] The *socket* is already connected.

45122 [ENOTSOCK] The *socket* argument does not refer to a socket.

45123 [EOPNOTSUPP] The socket protocol does not support *listen*( ).

45124 The *listen*( ) function may fail if:

45125 [EACCES] The calling process does not have appropriate privileges.

45126 [EINVAL] The *socket* has been shut down.

45127 [ENOBUFS] Insufficient resources are available in the system to complete the call.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

accept( ), *connect*( ), *socket*( )

XBD **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The DESCRIPTION is updated to describe the relationship of SOMAXCONN and the *backlog* argument.

45143 **NAME**

45144 llabs — return a long integer absolute value

45145 **SYNOPSIS**

45146 `#include <stdlib.h>`

45147 `long long llabs(long long i);`

45148 **DESCRIPTION**

45149 Refer to *labs( )*.

45150 **NAME**

45151 lldiv — compute quotient and remainder of a long division

45152 **SYNOPSIS**

45153 `#include <stdlib.h>`

45154 `lldiv_t lldiv(long long` *`numer`*`, long long` *`denom`*`);`

45155 **DESCRIPTION**

45156 Refer to *ldiv*( ).

NAME

llrint, llrintf, llrintl — round to the nearest integer value using current rounding direction

SYNOPSIS

```
#include <math.h>

long long llrint(double x);
long long llrintf(float x);
long long llrintl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value, rounding according to the current rounding direction.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value.

MX If *x* is NaN, a domain error shall occur, and an unspecified value is returned.

If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

If *x* is –Inf, a domain error shall occur and an unspecified value is returned.

If the correct value is positive and too large to represent as a **long long**, an unspecified value shall be returned. MX On systems that support the IEC 60559 Floating-Point option, a domain error shall occur; CX otherwise, a domain error may occur.

If the correct value is negative and too large to represent as a **long long**, an unspecified value shall be returned. MX On systems that support the IEC 60559 Floating-Point option, a domain error shall occur; CX otherwise, a domain error may occur.

ERRORS

These functions shall fail if:

MX

Domain Error — The *x* argument is NaN or ±Inf, or the correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Domain Error — The correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions provide floating-to-integer conversions. They round according to the current rounding direction. If the rounded value is outside the range of the return type, the numeric result is unspecified and the invalid floating-point exception is raised. When they raise no other floating-point exception and the result differs from the argument, they raise the inexact floating-point exception.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept( )*, *lrint( )*

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #53 is applied.

NAME

llround, llroundf, llroundl — round to nearest integer value

SYNOPSIS

```
#include <math.h>

long long llround(double x);
long long llroundf(float x);
long long llroundl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value, rounding halfway cases away from zero, regardless of the current rounding direction.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value.

MX If *x* is NaN, a domain error shall occur, and an unspecified value is returned.

If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

If *x* is –Inf, a domain error shall occur and an unspecified value is returned.

If the correct value is positive and too large to represent as a **long long**, an unspecified value shall be returned. MX On systems that support the IEC 60559 Floating-Point option, a domain error shall occur; CX otherwise, a domain error may occur.

If the correct value is negative and too large to represent as a **long long**, an unspecified value shall be returned. MX On systems that support the IEC 60559 Floating-Point option, a domain error shall occur; CX otherwise, a domain error may occur.

ERRORS

These functions shall fail if:

MX Domain Error — The *x* argument is NaN or ±Inf, or the correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Domain Error — The correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions differ from the *llrint*( ) functions in that the default rounding direction for the *llround*( ) functions round halfway cases away from zero and need not raise the inexact floating-point exception for non-integer arguments that round to within the range of the return type.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *lround*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #54 (SD5-XSH-ERN-75) is applied.

NAME

localeconv — return locale-specific information

SYNOPSIS

```
#include <locale.h>

struct lconv *localeconv(void);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *localeconv*( ) function shall set the components of an object with the type **struct lconv** with the values appropriate for the formatting of numeric quantities (monetary and otherwise) according to the rules of the current locale.

The members of the structure with type **char *** are pointers to strings, any of which (except **decimal_point**) can point to **""**, to indicate that the value is not available in the current locale or is of zero length. The members with type **char** are non-negative numbers, any of which can be {CHAR_MAX} to indicate that the value is not available in the current locale.

The members include the following:

char *decimal_point
: The radix character used to format non-monetary quantities.

char *thousands_sep
: The character used to separate groups of digits before the decimal-point character in formatted non-monetary quantities.

char *grouping
: A string whose elements taken as one-byte integer values indicate the size of each group of digits in formatted non-monetary quantities.

char *int_curr_symbol
: The international currency symbol applicable to the current locale. The first three characters contain the alphabetic international currency symbol in accordance with those specified in the ISO 4217:2015 standard. The fourth character (immediately preceding the null byte) is the character used to separate the international currency symbol from the monetary quantity.

char *currency_symbol
: The local currency symbol applicable to the current locale.

char *mon_decimal_point
: The radix character used to format monetary quantities.

char *mon_thousands_sep
: The separator for groups of digits before the decimal-point in formatted monetary quantities.

char *mon_grouping
: A string whose elements taken as one-byte integer values indicate the size of each group of digits in formatted monetary quantities.

char *positive_sign
: The string used to indicate a non-negative valued formatted monetary quantity.

45322 **char *negative_sign**
45323 The string used to indicate a negative valued formatted monetary quantity.

45324 **char int_frac_digits**
45325 The number of fractional digits (those after the decimal-point) to be displayed in an
45326 internationally formatted monetary quantity.

45327 **char frac_digits**
45328 The number of fractional digits (those after the decimal-point) to be displayed in a
45329 formatted monetary quantity.

45330 **char p_cs_precedes**
45331 Set to 1 if the **currency_symbol** precedes the value for a non-negative formatted monetary
45332 quantity. Set to 0 if the symbol succeeds the value.

45333 **char p_sep_by_space**
45334 Set to a value indicating the separation of the **currency_symbol**, the sign string, and the
45335 value for a non-negative formatted monetary quantity.

45336 **char n_cs_precedes**
45337 Set to 1 if the **currency_symbol** precedes the value for a negative formatted monetary
45338 quantity. Set to 0 if the symbol succeeds the value.

45339 **char n_sep_by_space**
45340 Set to a value indicating the separation of the **currency_symbol**, the sign string, and the
45341 value for a negative formatted monetary quantity.

45342 **char p_sign_posn**
45343 Set to a value indicating the positioning of the **positive_sign** for a non-negative formatted
45344 monetary quantity.

45345 **char n_sign_posn**
45346 Set to a value indicating the positioning of the **negative_sign** for a negative formatted
45347 monetary quantity.

45348 **char int_p_cs_precedes**
45349 Set to 1 or 0 if the **int_curr_symbol** respectively precedes or succeeds the value for a non-
45350 negative internationally formatted monetary quantity.

45351 **char int_n_cs_precedes**
45352 Set to 1 or 0 if the **int_curr_symbol** respectively precedes or succeeds the value for a
45353 negative internationally formatted monetary quantity.

45354 **char int_p_sep_by_space**
45355 Set to a value indicating the separation of the **int_curr_symbol**, the sign string, and the
45356 value for a non-negative internationally formatted monetary quantity.

45357 **char int_n_sep_by_space**
45358 Set to a value indicating the separation of the **int_curr_symbol**, the sign string, and the
45359 value for a negative internationally formatted monetary quantity.

45360 **char int_p_sign_posn**
45361 Set to a value indicating the positioning of the **positive_sign** for a non-negative
45362 internationally formatted monetary quantity.

45363 **char int_n_sign_posn**
45364 Set to a value indicating the positioning of the **negative_sign** for a negative internationally
45365 formatted monetary quantity.

The elements of **grouping** and **mon_grouping** are interpreted according to the following:

{CHAR_MAX} No further grouping is to be performed.

0 The previous element is to be repeatedly used for the remainder of the digits.

other The integer value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group.

The values of **p_sep_by_space**, **n_sep_by_space**, **int_p_sep_by_space**, and **int_n_sep_by_space** are interpreted according to the following:

0 No space separates the currency symbol and value.

1 If the currency symbol and sign string are adjacent, a space separates them from the value; otherwise, a space separates the currency symbol from the value.

2 If the currency symbol and sign string are adjacent, a space separates them; otherwise, a space separates the sign string from the value.

For **int_p_sep_by_space** and **int_n_sep_by_space**, the fourth character of **int_curr_symbol** is used instead of a space.

The values of **p_sign_posn**, **n_sign_posn**, **int_p_sign_posn**, and **int_n_sign_posn** are interpreted according to the following:

0 Parentheses surround the quantity and **currency_symbol** or **int_curr_symbol**.

1 The sign string precedes the quantity and **currency_symbol** or **int_curr_symbol**.

2 The sign string succeeds the quantity and **currency_symbol** or **int_curr_symbol**.

3 The sign string immediately precedes the **currency_symbol** or **int_curr_symbol**.

4 The sign string immediately succeeds the **currency_symbol** or **int_curr_symbol**.

The implementation shall behave as if no function in this volume of POSIX.1-2024 calls *localeconv*( ).

The *localeconv*( ) function need not be thread-safe; however, *localeconv*( ) shall avoid data races with all other functions.

RETURN VALUE

The *localeconv*( ) function shall return a pointer to the filled-in object. The application shall not CX modify the structure to which the return value points, nor any storage areas pointed to by pointers within the structure. The returned pointer, and pointers within the structure, might be CX invalidated or the structure or the storage areas might be overwritten by a subsequent call to CX *localeconv*( ). In addition, the returned pointer, and pointers within the structure, might be CX invalidated or the structure or the storage areas might be overwritten by subsequent calls to CX *setlocale*( ) with the categories LC_ALL, LC_MONETARY, or LC_NUMERIC, or by calls to *uselocale*( ) which change the categories LC_MONETARY or LC_NUMERIC. The returned pointer, pointers within the structure, the structure, and the storage areas might also be invalidated if the calling thread is terminated.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The following table illustrates the rules which may be used by four countries to format monetary quantities.

Country	Positive Format	Negative Format	International Format
Italy	€.1.230	–€.1.230	EUR.1.230
Netherlands	€ 1.234,56	€ –1.234,56	EUR 1.234,56
Norway	kr1.234,56	kr1.234,56–	NOK 1.234,56
Switzerland	SFrs.1,234.56	SFrs.1,234.56C	CHF 1,234.56

For these four countries, the respective values for the monetary members of the structure returned by *localeconv*( ) are:

	Italy	Netherlands	Norway	Switzerland
int_curr_symbol	`"EUR."`	`"EUR "`	`"NOK "`	`"CHF "`
currency_symbol	`"€."`	`"€"`	`"kr"`	`"SFrs."`
mon_decimal_point	`""`	`","`	`","`	`"."`
mon_thousands_sep	`"."`	`"."`	`"."`	`","`
mon_grouping	`"\3"`	`"\3"`	`"\3"`	`"\3"`
positive_sign	`""`	`""`	`""`	`""`
negative_sign	`"-"`	`"-"`	`"-"`	`"C"`
int_frac_digits	0	2	2	2
frac_digits	0	2	2	2
p_cs_precedes	1	1	1	1
p_sep_by_space	0	1	0	0
n_cs_precedes	1	1	1	1
n_sep_by_space	0	1	0	0
p_sign_posn	1	1	1	1
n_sign_posn	1	4	2	2
int_p_cs_precedes	1	1	1	1
int_n_cs_precedes	1	1	1	1
int_p_sep_by_space	0	0	0	0
int_n_sep_by_space	0	0	0	0
int_p_sign_posn	1	1	1	1
int_n_sign_posn	1	4	4	2

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fprintf( ), *fscanf*( ), *isalpha*( ), *nl_langinfo*( ), *setlocale*( ), *strcat*( ), *strchr*( ), *strcmp*( ), *strcoll*( ), *strcpy*( ), *strftime*( ), *strlen*( ), *strpbrk*( ), *strspn*( ), *strtok*( ), *strxfrm*( ), *strtod*( ), *uselocale*( )

XBD **<langinfo.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ANSI C standard.

Issue 6

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

The RETURN VALUE section is rewritten to avoid use of the term ``must''.

This reference page is updated for alignment with the ISO/IEC 9899: 1999 standard.

ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/31 is applied, removing references to **int_curr_symbol** and updating the descriptions of **p_sep_by_space** and **n_sep_by_space**. These changes are for alignment with the ISO C standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

The definitions of **int_curr_symbol** and **currency_symbol** are updated.

The examples in the APPLICATION USAGE section are updated.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0362 [75] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0200 [656] is applied.

Issue 8

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899: 2018 standard.

NAME

localtime, localtime_r — convert a time value to a broken-down local time

SYNOPSIS

```
#include <time.h>

struct tm *localtime(const time_t *timer);
CX struct tm *localtime_r(const time_t *restrict timer,
        struct tm *restrict result);
```

DESCRIPTION

CX For *localtime*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *localtime*( ) function shall convert the time in seconds since the Epoch pointed to by *timer* into a broken-down time, expressed as a local time. The function corrects for the timezone and any seasonal time adjustments. CX Local timezone information shall be set as though *localtime*( ) calls *tzset*( ).

The relationship between a time in seconds since the Epoch used as an argument to *localtime*( ) and the **tm** structure (defined in the **<time.h>** header) is that the result shall be as specified in the expression given in the definition of seconds since the Epoch (see XBD Section 4.19, on page 107) corrected for timezone and any seasonal time adjustments, where the names in the structure and in the expression correspond.

The same relationship shall apply for *localtime_r*( ).

The *localtime*( ) function need not be thread-safe; however, *localtime*( ) shall avoid data races with all functions other than itself, *asctime*( ), *ctime*( ), and *gmtime*( ).

The *asctime*( ), *ctime*( ), *gmtime*( ), and *localtime*( ) functions shall return values in one of two static objects: a broken-down time structure and an array of type **char**. Execution of any of the functions that return a pointer to one of these object types may overwrite the information in any object of the same type pointed to by the value returned from any previous call to any of them.

CX The *localtime_r*( ) function shall convert the time in seconds since the Epoch pointed to by *timer* into a broken-down time stored in the structure to which *result* points. The *localtime_r*( ) function shall also return a pointer to that same structure.

Unlike *localtime*( ), the *localtime_r*( ) function is not required to set *tzname*. If *localtime_r*( ) sets *tzname*, it shall also set *daylight* and *timezone*. If *localtime_r*( ) does not set *tzname*, it shall not set *daylight* and shall not set *timezone*. If the **tm** structure member *tm_zone* is accessed after the value of *TZ* is subsequently modified, the behaviour is undefined.

RETURN VALUE

Upon successful completion, the *localtime*( ) function shall return a pointer to the broken-down CX time structure. If an error is detected, *localtime*( ) shall return a null pointer and set *errno* to indicate the error.

Upon successful completion, *localtime_r*( ) shall return a pointer to the structure pointed to by the argument *result*. If an error is detected, *localtime_r*( ) shall return a null pointer and set *errno* to indicate the error.

ERRORS

CX The *localtime*( ) and *localtime_r*( ) functions shall fail if:

CX [EOVERFLOW] The result cannot be represented.

EXAMPLES

Getting the Local Date and Time

The following example uses the *time*( ) function to calculate the time elapsed, in seconds, since January 1, 1970 0:00 UTC (the Epoch), *localtime*( ) to convert that value to a broken-down time, and *asctime*( ) to convert the broken-down time values into a printable string.

```
#include <stdio.h>
#include <time.h>

int main(void)
{
    time_t result;

    result = time(NULL);
    printf("%s%ju secs since the Epoch\n",
        asctime(localtime(&result)),
            (uintmax_t)result);
    return(0);
}
```

This example writes the current time to *stdout* in a form like this:

```
Wed Jun 26 10:32:15 1996
835810335 secs since the Epoch
```

Getting the Modification Time for a File

The following example prints the last data modification timestamp in the local timezone for a given file.

```
#include <stdio.h>
#include <time.h>
#include <sys/stat.h>

int
print_file_time(const char *pathname)
{
    struct stat statbuf;
    struct tm *tm;
    char timestr[BUFSIZ];

    if(stat(pathname, &statbuf) == -1)
        return -1;
    if((tm = localtime(&statbuf.st_mtime)) == NULL)
        return -1;
    if(strftime(timestr, sizeof(timestr), "%Y-%m-%d %H:%M:%S", tm) == 0)
        return -1;
    printf("%s: %s.%09ld\n", pathname, timestr, statbuf.st_mtim.tv_nsec);
    return 0;
}
```


45550 **Timing an Event**

45551 The following example gets the current time, converts it to a string using *localtime*( ) and
45552 *asctime*( ), and prints it to standard output using *fputs*( ). It then prints the number of minutes to
45553 an event being timed.

```
45554 #include <time.h>
45555 #include <stdio.h>
45556 ...
45557 time_t now;
45558 int minutes_to_event;
45559 ...
45560 time(&now);
45561 printf("The time is ");
45562 fputs(asctime(localtime(&now)), stdout);
45563 printf("There are still %d minutes to the event.\n",
45564     minutes_to_event);
45565 ...
```

45566 **APPLICATION USAGE**

45567 The *localtime_r*( ) function is thread-safe and returns values in a user-supplied buffer instead of
45568 possibly using a static data area that may be overwritten by each call.

45569 **RATIONALE**

45570 None.

45571 **FUTURE DIRECTIONS**

45572 None.

45573 **SEE ALSO**

45574 *asctime*( ), *clock*( ), *ctime*( ), *difftime*( ), *futimens*( ), *getdate*( ), *gmtime*( ), *mktime*( ), *strftime*( ),
45575 *strptime*( ), *time*( ), *tzset*( )

45576 XBD Section 4.19 (on page 107), **<time.h>**

45577 **CHANGE HISTORY**

45578 First released in Issue 1. Derived from Issue 1 of the SVID.

45579 **Issue 5**

45580 A note indicating that the *localtime*( ) function need not be reentrant is added to the
45581 DESCRIPTION.

45582 The *localtime_r*( ) function is included for alignment with the POSIX Threads Extension.

45583 **Issue 6**

45584 The *localtime_r*( ) function is marked as part of the Thread-Safe Functions option.

45585 Extensions beyond the ISO C standard are marked.

45586 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
45587 its avoidance of possibly using a static data area.

45588 The **restrict** keyword is added to the *localtime_r*( ) prototype for alignment with the
45589 ISO/IEC 9899:1999 standard.

45590 Examples are added.

45591 IEEE Std 1003.1-2001/Cor 1-2002, itemm XSH/TC1/D6/32 is applied, adding the
45592 [EOVERFLOW] error.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/55 is applied, updating the error handling for *localtime_r*( ).

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/56 is applied, adding a requirement that if *localtime_r*( ) does not set the *tzname* variable, it shall not set the *daylight* or *timezone* variables. On systems supporting XSI, the *daylight*, *timezone*, and *tzname* variables should all be set to provide information for the same timezone. This updates the description of *localtime_r*( ) to mention *daylight* and *timezone* as well as *tzname*. The SEE ALSO section is updated.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

The *localtime_r*( ) function is moved from the Thread-Safe Functions option to the Base.

Changes are made to the EXAMPLES section related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0363 [291] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0201 [664] is applied.

Issue 8

Austin Group Defect 1125 is applied, changing ``Local timezone information is used'' to ``Local timezone information shall be set''.

Austin Group Defect 1302 is applied, aligning the *localtime*( ) function with the ISO/IEC 9899:2018 standard.

Austin Group Defect 1376 is applied, removing CX shading from some text derived from the ISO C standard and updating it to match the ISO C standard.

Austin Group Defect 1533 is applied, adding *tm_gmtoff* and *tm_zone* to the **tm** structure.

Austin Group Defect 1570 is applied, removing extra spacing in `"=="`.

NAME

lockf — record locking on files

SYNOPSIS

XSI
```
#include <unistd.h>

int lockf(int fildes, int function, off_t size);
```

DESCRIPTION

The *lockf*( ) function shall lock sections of a file with advisory-mode process-owned file locks. Calls to *lockf*( ) from threads in other processes which attempt to lock the locked file section shall either return an error value or block until the section becomes unlocked. All the locks for a process are removed when the process terminates. Record locking with *lockf*( ) shall be supported for regular files and may be supported for other files.

The *fildes* argument is an open file descriptor. To establish a lock with this function, the file descriptor shall be opened with write-only permission (O_WRONLY) or with read/write permission (O_RDWR).

The *function* argument is a control value which specifies the action to be taken. The permissible values for *function* are defined in **<unistd.h>** as follows:

Function	Description
F_ULOCK	Unlock locked sections.
F_LOCK	Lock a section for exclusive use.
F_TLOCK	Test and lock a section for exclusive use.
F_TEST	Test a section for locks by other processes.

F_TEST shall detect if a lock by another process is present on the specified section.

F_LOCK and F_TLOCK shall both lock a section of a file if the section is available.

F_ULOCK shall remove locks from a section of the file.

The *size* argument is the number of contiguous bytes to be locked or unlocked. The section to be locked or unlocked starts at the current offset in the file and extends forward for a positive size or backward for a negative size (the preceding bytes up to but not including the current offset). If *size* is 0, the section from the current offset through the largest possible file offset shall be locked (that is, from the current offset through the present or any future end-of-file). An area need not be allocated to the file to be locked because locks may exist past the end-of-file.

The sections locked with F_LOCK or F_TLOCK may, in whole or in part, contain or be contained by a previously locked section for the same process. When this occurs, or if adjacent locked sections would occur, the sections shall be combined into a single locked section. If the request would cause the number of locks to exceed a system-imposed limit, the request shall fail.

F_LOCK and F_TLOCK requests differ only by the action taken if the section is not available. F_LOCK shall block the calling thread until the section is available. F_TLOCK shall cause the function to fail if the section is already locked by another process.

Process-owned file locks shall be released on first close by the locking process of any file descriptor for the file.

F_ULOCK requests may release (wholly or in part) one or more locked sections controlled by the process. Locked sections shall be unlocked starting at the current file offset through *size* bytes or to the end-of-file if *size* is (**off_t**)0. When all of a locked section is not released (that is, when the beginning or end of the area to be unlocked falls within a locked section), the remaining portions of that section shall remain locked by the process. Releasing the center portion of a locked

section shall cause the remaining locked beginning and end portions to become two separate locked sections. If the request would cause the number of locks in the system to exceed a system-imposed limit, the request shall fail.

A potential for deadlock occurs if the threads of a process controlling a locked section are blocked by accessing a locked section of another process. If the system detects that deadlock would occur, *lockf*( ) shall fail with an [EDEADLK] error.

The interaction between *fcntl*( ) and *lockf*( ) locks is unspecified.

Blocking on a section shall be interrupted by any signal.

An F_ULOCK request in which *size* is non-zero and the offset of the last byte of the requested section is the maximum value for an object of type **off_t**, when the process has an existing lock in which *size* is 0 and which includes the last byte of the requested section, shall be treated as a request to unlock from the start of the requested section with a size equal to 0. Otherwise, an F_ULOCK request shall attempt to unlock only the requested section.

Attempting to lock a section of a file that is associated with a buffered stream produces unspecified results.

RETURN VALUE

Upon successful completion, *lockf*( ) shall return 0. Otherwise, it shall return –1, set *errno* to indicate an error, and existing locks shall not be changed.

ERRORS

The *lockf*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid open file descriptor; or *function* is F_LOCK or F_TLOCK and *fildes* is not a valid file descriptor open for writing.

[EACCES] or [EAGAIN]
The *function* argument is F_TLOCK or F_TEST and the section is already locked by another process.

[EDEADLK] The *function* argument is F_LOCK and a deadlock is detected.

[EINTR] A signal was caught during execution of the function.

[EINVAL] The *function* argument is not one of F_LOCK, F_TLOCK, F_TEST, or F_ULOCK; or *size* plus the current file offset is less than 0.

[EOVERFLOW] The offset of the first, or if *size* is not 0 then the last, byte in the requested section cannot be represented correctly in an object of type **off_t**.

The *lockf*( ) function may fail if:

[EAGAIN] The *function* argument is F_LOCK or F_TLOCK and the file is mapped with *mmap*( ).

[EDEADLK] or [ENOLCK]
The *function* argument is F_LOCK, F_TLOCK, or F_ULOCK, and the request would cause the number of locks to exceed a system-imposed limit.

[EOPNOTSUPP] or [EINVAL]
The implementation does not support the locking of files of the type indicated by the *fildes* argument.

EXAMPLES

Locking a Portion of a File

In the following example, a file named **/home/cnd/mod1** is being modified. Other processes that use locking are prevented from changing it during this process. Only the first 10 000 bytes are locked, and the lock call fails if another process has any part of this area locked already.

```
#include <fcntl.h>
#include <unistd.h>

int fildes;
int status;
...
fildes = open("/home/cnd/mod1", O_RDWR);
status = lockf(fildes, F_TLOCK, (off_t)10000);
```

APPLICATION USAGE

Record-locking should not be used in combination with buffered standard I/O streams (see Section 2.5, on page 521). Instead, non-buffered I/O should be used. Unexpected results may occur in processes that do buffering in the user address space. The process may later read/write data which is/was locked. Functions that operate on standard I/O streams are the most common source of such buffering.

The *alarm*( ) function may be used to provide a timeout facility in applications requiring it.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *chmod*( ), *close*( ), *creat*( ), *fcntl*( ), *fopen*( ), *mmap*( ), *open*( ), *read*( ), *write*( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Large File Summit extensions are added. In particular, the description of [EINVAL] is clarified and moved from optional to mandatory status.

A note is added to the DESCRIPTION indicating the effects of attempting to lock a section of a file that is associated with a buffered stream.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #054 is applied, updating the DESCRIPTION.

Issue 8

Austin Group Defect 768 is applied, adding OFD-owned file locks.

Austin Group Defect 1672 is applied, changing the APPLICATION USAGE section.

NAME

log, logf, logl — natural logarithm function

SYNOPSIS

```
#include <math.h>

double log(double x);
float logf(float x);
long double logl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the natural logarithm of their argument x, $\log_e(x)$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the natural logarithm of x.

If x is ±0, a pole error shall occur and *log*( ), *logf*( ), and *logl*( ) shall return –HUGE_VAL, –HUGE_VALF, and –HUGE_VALL, respectively.

MX For finite values of x that are less than 0, or if x is –Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

MX If x is NaN, a NaN shall be returned.

If x is 1, +0 shall be returned.

If x is +Inf, x shall be returned.

ERRORS

These functions shall fail if:

MX Domain Error — The finite value of x is negative, or x is –Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error — The value of x is zero.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exp( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *log10*( ), *log1p*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

The *logf*( ) and *logl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

NAME

log10, log10f, log10l — base 10 logarithm function

SYNOPSIS

```
#include <math.h>

double log10(double x);
float log10f(float x);
long double log10l(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the base 10 logarithm of their argument x, $\log_{10}(x)$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the base 10 logarithm of x.

If x is ±0, a pole error shall occur and *log10*( ), *log10f*( ), and *log10l*( ) shall return –HUGE_VAL, –HUGE_VALF, and –HUGE_VALL, respectively.

MX For finite values of x that are less than 0, or if x is –Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

MX If x is NaN, a NaN shall be returned.

If x is 1, +0 shall be returned.

If x is +Inf, +Inf shall be returned.

ERRORS

These functions shall fail if:

MX Domain Error — The finite value of x is negative, or x is –Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error — The value of x is zero.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( ), *log*( ), *pow*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

The *log10f*( ) and *log10l*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

NAME

log1p, log1pf, log1pl — compute a natural logarithm

SYNOPSIS

```
#include <math.h>

double log1p(double x);
float log1pf(float x);
long double log1pl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute $\log_e(1.0 + x)$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the natural logarithm of $1.0 + x$.

If x is –1, a pole error shall occur and *log1p*( ), *log1pf*( ), and *log1pl*( ) shall return –HUGE_VAL, –HUGE_VALF, and –HUGE_VALL, respectively.

MX For finite values of x that are less than –1, or if x is –Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

MX If x is NaN, a NaN shall be returned.

If x is ±0, or +Inf, x shall be returned.

MXX If x is subnormal, x should be returned.

MX If x is subnormal, a range error may occur and, if x is not returned, *log1p*( ), *log1pf*( ), and *log1pl*( ) shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

ERRORS

These functions shall fail if:

MX Domain Error — The finite value of x is less than –1, or x is –Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error — The value of x is –1.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

These functions may fail if:

MX

Range Error	The value of *x* is subnormal.
	If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *log*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

The *log1p*( ) function is no longer marked as an extension.

The *log1pf*( ) and *log1pl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0364 [68] is applied.

Issue 8

Austin Group Defect 1382 is applied, rearranging the text describing the behavior when *x* is subnormal to avoid the need for two shading changes.

NAME

log2, log2f, log2l — compute base 2 logarithm functions

SYNOPSIS

```
#include <math.h>

double log2(double x);
float log2f(float x);
long double log2l(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the base 2 logarithm of their argument x, $\log_2(x)$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the base 2 logarithm of x.

If x is ±0, a pole error shall occur and *log2*( ), *log2f*( ), and *log2l*( ) shall return –HUGE_VAL, –HUGE_VALF, and –HUGE_VALL, respectively.

MX For finite values of x that are less than 0, or if x is –Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

MX If x is NaN, a NaN shall be returned.

If x is 1, +0 shall be returned.

If x is +Inf, x shall be returned.

ERRORS

These functions shall fail if:

MX Domain Error — The finite value of x is less than zero, or x is –Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error — The value of x is zero.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *log*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

NAME

logb, logbf, logbl — radix-independent exponent

SYNOPSIS

```
#include <math.h>

double logb(double x);
float logbf(float x);
long double logbl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the exponent of x, which is the integral part of $\log_r |x|$, as a signed floating-point value, for non-zero x, where r is the radix of the machine's floating-point arithmetic, which is the value of FLT_RADIX defined in the **<float.h>** header.

If x is subnormal it is treated as though it were normalized; thus for finite positive x:

$$1 <= x * \mathrm{FLT\_RADIX}^{-\mathrm{logb}(x)} < \mathrm{FLT\_RADIX}$$

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the exponent of x.

MX The returned value shall be exact and shall be independent of the current rounding direction mode.

If x is ±0, *logb*(), *logbf*(), and *logbl*() shall return –HUGE_VAL, –HUGE_VALF, and –HUGE_VALL, respectively.

MX On systems that support the IEC 60559 Floating-Point option, a pole error shall occur;
CX otherwise, a pole error may occur.

MX If x is NaN, a NaN shall be returned.

MX If x is ±Inf, +Inf shall be returned.

ERRORS

These functions shall fail if:

MX Pole Error — The value of x is ±0.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

These functions may fail if:

Pole Error — The value of x is 0.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *ilogb*( ), *scalbln*( )

XBD Section 4.23 (on page 109), **<float.h>**, **<math.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The *logb*( ) function is no longer marked as an extension.

The *logbf*( ) and *logbl*( ) functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

Issue 7

ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #50 (SD5-XSH-ERN-76) is applied.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

46061 **NAME**

46062 logf, logl — natural logarithm function

46063 **SYNOPSIS**

46064 `#include <math.h>`

46065 `float logf(float x);`

46066 `long double logl(long double x);`

46067 **DESCRIPTION**

46068 Refer to *log( )*.

NAME

longjmp — non-local goto

SYNOPSIS

```
#include <setjmp.h>

_Noreturn void longjmp(jmp_buf env, int val);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *longjmp*( ) function shall restore the environment saved by the most recent invocation of *setjmp*( ) in the same process, with the corresponding **jmp_buf** argument. If the most recent invocation of *setjmp*( ) with the corresponding **jmp_buf** occurred in another thread, or if there is no such invocation, or if the function containing the invocation of *setjmp*( ) has terminated execution in the interim, or if the invocation of *setjmp*( ) was within the scope of an identifier with variably modified type and execution has left that scope in the interim, the behavior is CX undefined. It is unspecified whether *longjmp*( ) restores the signal mask, leaves the signal mask unchanged, or restores it to its value at the time *setjmp*( ) was called.

All accessible objects have values, and all other components of the abstract machine have state (for example, floating-point status flags and open files), as of the time *longjmp*( ) was called, except that the values of objects of automatic storage duration are unspecified if they meet all the following conditions:

- They are local to the function containing the corresponding *setjmp*( ) invocation.
- They do not have volatile-qualified type.
- They are changed between the *setjmp*( ) invocation and *longjmp*( ) call.

CX Although *longjmp*( ) is an async-signal-safe function, if it is invoked from a signal handler which interrupted a non-async-signal-safe function or equivalent (such as the processing equivalent to *exit*( ) performed after a return from the initial call to *main*( )), the behavior of any subsequent call to a non-async-signal-safe function or equivalent is undefined.

The effect of a call to *longjmp*( ) where initialization of the **jmp_buf** structure was not performed in the calling thread is undefined.

RETURN VALUE

After *longjmp*( ) is completed, thread execution shall continue as if the corresponding invocation of *setjmp*( ) had just returned the value specified by *val*. The *longjmp*( ) function shall not cause *setjmp*( ) to return 0; if *val* is 0, *setjmp*( ) shall return 1.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Applications whose behavior depends on the value of the signal mask should not use *longjmp*( ) and *setjmp*( ), since their effect on the signal mask is unspecified, but should instead use the *siglongjmp*( ) and *sigsetjmp*( ) functions (which can save and restore the signal mask under application control).

It is recommended that applications do not call *longjmp*( ) or *siglongjmp*( ) from signal handlers. To avoid undefined behavior when calling these functions from a signal handler, the application

needs to ensure one of the following two things:

1. After the call to *longjmp*( ) or *siglongjmp*( ) the process only calls async-signal-safe functions and does not return from the initial call to *main*( ).
2. Any signal whose handler calls *longjmp*( ) or *siglongjmp*( ) is blocked during *every* call to a non-async-signal-safe function, and no such calls are made after returning from the initial call to *main*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setjmp( ), *sigaction*( ), *siglongjmp*( ), *sigsetjmp*( )

XBD **<setjmp.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION now explicitly makes *longjmp*( )'s effect on the signal mask unspecified.

The DESCRIPTION is updated for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0365 [394] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0202 [516] is applied.

Issue 8

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899: 2018 standard.

NAME

lrand48 — generate uniformly distributed pseudo-random non-negative long integers

SYNOPSIS

XSI

```
#include <stdlib.h>

long lrand48(void);
```

DESCRIPTION

Refer to *drand48( )*.

NAME

lrint, lrintf, lrintl — round to nearest integer value using current rounding direction

SYNOPSIS

```
#include <math.h>

long lrint(double x);
long lrintf(float x);
long lrintl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value, rounding according to the current rounding direction.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value.

MX If *x* is NaN, a domain error shall occur and an unspecified value is returned.

If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

If *x* is –Inf, a domain error shall occur and an unspecified value is returned.

If the correct value is positive and too large to represent as a **long**, an unspecified value shall be returned. MX On systems that support the IEC 60559 Floating-Point option, a domain error shall occur; CX otherwise, a domain error may occur.

If the correct value is negative and too large to represent as a **long**, an unspecified value shall be returned. MX On systems that support the IEC 60559 Floating-Point option, a domain error shall occur; CX otherwise, a domain error may occur.

ERRORS

These functions shall fail if:

MX Domain Error — The *x* argument is NaN or ±Inf, or the correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Domain Error — The correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions provide floating-to-integer conversions. They round according to the current rounding direction. If the rounded value is outside the range of the return type, the numeric result is unspecified and the invalid floating-point exception is raised. When they raise no other floating-point exception and the result differs from the argument, they raise the inexact floating-point exception.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *llrint*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #53 (SD5-XSH-ERN-77) is applied.

NAME

lround, lroundf, lroundl — round to nearest integer value

SYNOPSIS

```
#include <math.h>

long lround(double x);
long lroundf(float x);
long lroundl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value, rounding halfway cases away from zero, regardless of the current rounding direction.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value.

MX If *x* is NaN, a domain error shall occur and an unspecified value is returned.

If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

If *x* is –Inf, a domain error shall occur and an unspecified value is returned.

If the correct value is positive and too large to represent as a **long**, an unspecified value shall be returned. MX On systems that support the IEC 60559 Floating-Point option, a domain shall occur; CX otherwise, a domain error may occur.

If the correct value is negative and too large to represent as a **long**, an unspecified value shall be returned. MX On systems that support the IEC 60559 Floating-Point option, a domain shall occur; CX otherwise, a domain error may occur.

ERRORS

These functions shall fail if:

MX Domain Error — The *x* argument is NaN or ±Inf, or the correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Domain Error — The correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions differ from the *lrint*( ) functions in the default rounding direction, with the *lround*( ) functions rounding halfway cases away from zero and needing not to raise the inexact floating-point exception for non-integer arguments that round to within the range of the return type.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *llround*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #54 (SD5-XSH-ERN-78) is applied.

NAME

lsearch, lfind — linear search and update

SYNOPSIS

XSI

```
#include <search.h>

void *lsearch(const void *key, void *base, size_t *nelp, size_t width,
    int (*compar)(const void *, const void *));
void *lfind(const void *key, const void *base, size_t *nelp,
    size_t width, int (*compar)(const void *, const void *));
```

DESCRIPTION

The *lsearch*( ) function shall linearly search the table and return a pointer into the table for the matching entry. If the entry does not occur, it shall be added at the end of the table. The *key* argument points to the entry to be sought in the table. The *base* argument points to the first element in the table. The *width* argument is the size of an element in bytes. The *nelp* argument points to an integer containing the current number of elements in the table. The integer to which *nelp* points shall be incremented if the entry is added to the table. The *compar* argument points to a comparison function which the application shall supply (for example, *strcmp*( )). It is called with two arguments that point to the elements being compared. The application shall ensure that the function returns 0 if the elements are equal, and non-zero otherwise.

The *lfind*( ) function shall be equivalent to *lsearch*( ), except that if the entry is not found, it is not added to the table. Instead, a null pointer is returned.

RETURN VALUE

If the searched for entry is found, both *lsearch*( ) and *lfind*( ) shall return a pointer to it. Otherwise, *lfind*( ) shall return a null pointer and *lsearch*( ) shall return a pointer to the newly added element.

Both functions shall return a null pointer in case of error.

ERRORS

No errors are defined.

EXAMPLES

Storing Strings in a Table

This fragment reads in less than or equal to TABSIZE strings of length less than or equal to ELSIZE and stores them in a table, eliminating duplicates.

```
#include <stdio.h>
#include <string.h>
#include <search.h>

#define TABSIZE 50
#define ELSIZE 120

...
    char line[ELSIZE], tab[TABSIZE][ELSIZE];
    size_t nel = 0;
    ...
    while (fgets(line, ELSIZE, stdin) != NULL && nel < TABSIZE)
        (void) lsearch(line, tab, &nel,
            ELSIZE, (int (*)(const void *, const void *)) strcmp);
    ...
```


Finding a Matching Entry

The following example finds any line that reads "This is a test.".

```
#include <search.h>
#include <string.h>
...
char line[ELSIZE], tab[TABSIZE][ELSIZE];
size_t nel = 0;
char *findline;
void *entry;

findline = "This is a test.\n";

entry = lfind(findline, tab, &nel, ELSIZE, (
    int (*)(const void *, const void *)) strcmp);
```

APPLICATION USAGE

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

Undefined results can occur if there is not enough room in the table to add a new item.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

hcreate( ), *tdelete*( )

XBD **<search.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

NAME

lseek — move the read/write file offset

SYNOPSIS

```
#include <unistd.h>

off_t lseek(int fildes, off_t offset, int whence);
```

DESCRIPTION

The *lseek*( ) function shall set the file offset for the open file description associated with the file descriptor *fildes*, as follows:

- If *whence* is SEEK_SET, the file offset shall be set to *offset* bytes.
- If *whence* is SEEK_CUR, the file offset shall be set to its current location plus *offset*.
- If *whence* is SEEK_END, the file offset shall be set to the size of the file plus *offset*.
- If *whence* is SEEK_HOLE, the file offset shall be set to the smallest location of a byte within a hole and not less than *offset*, except that if *offset* falls beyond the last byte not within a hole, then the file offset may be set to the file size instead. It shall be an error if *offset* is greater than or equal to the size of the file.
- If *whence* is SEEK_DATA, the file offset shall be set to the smallest location of a byte not within a hole and not less than *offset*. It shall be an error if no such byte exists.

The symbolic constants SEEK_SET, SEEK_CUR, SEEK_END, SEEK_HOLE, and SEEK_DATA are defined in **<unistd.h>**.

A hole is a contiguous region of bytes within a file, all having the value of zero. Not all bytes with the value zero need belong to a hole; however, all seekable files shall have a virtual hole starting at the current size of the file, whether or not the file is sparse.

The behavior of *lseek*( ) on devices which are incapable of seeking is implementation-defined. The value of the file offset associated with such a device is undefined.

The *lseek*( ) function shall allow the file offset to be set beyond the end of the existing data in the file. If data is later written at this point, subsequent reads of data in the gap shall return bytes with the value 0 until data is actually written into the gap.

The *lseek*( ) function shall not, by itself, extend the size of a file.

SHM If *fildes* refers to a shared memory object, the result of the *lseek*( ) function is unspecified.

TYM If *fildes* refers to a typed memory object, the result of the *lseek*( ) function is unspecified.

RETURN VALUE

Upon successful completion, the resulting offset, as measured in bytes from the beginning of the file, shall be returned. Otherwise, –1 shall be returned, *errno* shall be set to indicate the error, and the file offset shall remain unchanged.

ERRORS

The *lseek*( ) function shall fail if:

[EBADF] The *fildes* argument is not an open file descriptor.

[EINVAL] The *whence* argument is not a proper value, or the resulting file offset would be negative for a regular file, block special file, or directory.

[ENXIO] The *whence* argument is SEEK_HOLE or SEEK_DATA, and *offset* is greater than or equal to the file size; or the *whence* argument is SEEK_DATA and the offset falls beyond the last byte not within a hole.

46387 [EOVERFLOW] The resulting file offset would be a value which cannot be represented
46388 correctly in an object of type **off_t**.

46389 [ESPIPE] The *fildes* argument is associated with a pipe, FIFO, or socket.

46390 **EXAMPLES**

46391 None.

46392 **APPLICATION USAGE**

46393 None.

46394 **RATIONALE**

46395 The ISO C standard includes the functions *fgetpos*( ) and *fsetpos*( ), which work on very large files
46396 by use of a special positioning type.

46397 Although *lseek*( ) may position the file offset beyond the end of the file, this function does not
46398 itself extend the size of the file. While the only function in POSIX.1-2024 that may directly extend
46399 the size of the file is *write*( ), *truncate*( ), and *ftruncate*( ), several functions originally derived from
46400 the ISO C standard, such as *fwrite*( ), *fprintf*( ), and so on, may do so (by causing calls on *write*( )).

46401 An invalid file offset that would cause [EINVAL] to be returned may be both implementation-
46402 defined and device-dependent (for example, memory may have few invalid values). A negative
46403 file offset may be valid for some devices in some implementations.

46404 The POSIX.1-1990 standard did not specifically prohibit *lseek*( ) from returning a negative offset.
46405 Therefore, an application was required to clear *errno* prior to the call and check *errno* upon return
46406 to determine whether a return value of (**off_t**)−1 is a negative offset or an indication of an error
46407 condition. The standard developers did not wish to require this action on the part of a
46408 conforming application, and chose to require that *errno* be set to [EINVAL] when the resulting
46409 file offset would be negative for a regular file, block special file, or directory.

46410 Not all file systems support holes, and even where sparse files are supported, not all contiguous
46411 blocks of zero bytes are required to be recognized as a hole. However, since all files are required
46412 to have a virtual hole starting at the current file size, application writers can use SEEK_HOLE
46413 and SEEK_DATA to optimize algorithms that can run faster when it is known that a block of
46414 bytes is all zeros, because a non-sparse file will correctly report the entire file as a single non-
46415 hole. A trivial recursive implementation for these two constants would be as follows, however,
46416 for file systems that support sparse files, implementations are encouraged to do better.

```
46417 off_t lseek(int fildes, off_t offset, int whence)
46418 {
46419     off_t cur, end;
46420     switch (whence)
46421     {
46422     case SEEK_HOLE:
46423     case SEEK_DATA:
46424         cur = lseek(fildes, 0, SEEK_CUR);
46425         if (cur < 0)
46426             return cur;
46427         end = lseek(fildes, 0, SEEK_END);
46428         if (end < 0)
46429             return end;
46430         if (offset < end)
46431             return whence == SEEK_HOLE ?
46432                 end : lseek(fildes, offset, SEEK_SET);
46433         lseek(fildes, cur, SEEK_SET);
```



```
46434                errno = ENXIO;
46435                return -1;
46436            default:
46437                ... /* Existing implementation */
46438            }
46439        }
```

Note that although the above looks like user-space code, *lseek*( ) cannot be implemented with recursive calls in user space because this would not conform to the atomicity requirements in Section 2.9.7 (on page 547).

FUTURE DIRECTIONS

None.

SEE ALSO

open( )

XBD **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

Large File Summit extensions are added.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [EOVERFLOW] error condition is added. This change is to support large files.

An additional [ESPIPE] error condition is added for sockets.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *lseek*( ) results are unspecified for typed memory objects.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0366 [421] is applied.

Issue 8

Austin Group Defect 415 is applied, adding SEEK_HOLE and SEEK_DATA.

NAME

lstat — get file status

SYNOPSIS

```
#include <sys/stat.h>

int lstat(const char *restrict path, struct stat *restrict buf);
```

DESCRIPTION

Refer to *fstatat( )*.

NAME

malloc — a memory allocator

SYNOPSIS

```
#include <stdlib.h>

void *malloc(size_t size);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *malloc*( ) function shall allocate unused space for an object whose size in bytes is specified by *size* and whose value is unspecified.

The order and contiguity of storage allocated by successive calls to *malloc*( ) is unspecified. The pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to a pointer to any type of object with a fundamental alignment requirement and then used to access such an object in the space allocated (until the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object disjoint from any other object. The pointer returned points to the start (lowest byte address) of the allocated space. If the space cannot be allocated, a null pointer shall be returned. If the size of the space requested is 0, the behavior is implementation-defined: either a null pointer shall be returned, or the behavior shall be as if the size were some non-zero value, except that the behavior is undefined if the returned pointer is used to access an object.

For purposes of determining the existence of a data race, *malloc*( ) shall behave as though it accessed only memory locations accessible through its argument and not other static duration storage. The function may, however, visibly modify the storage that it allocates. Calls to ADV *aligned_alloc*( ), *calloc*( ), *free*( ), *malloc*( ), *posix_memalign*( ), CX *reallocarray*( ), and *realloc*( ) that allocate or deallocate a particular region of memory shall occur in a single total order (see Section 4.15.1, on page 100), and each such deallocation call shall synchronize with the next allocation (if any) in this order.

RETURN VALUE

Upon successful completion, *malloc*( ) shall return a pointer to the allocated space; if *size* is 0, the application shall ensure that the pointer is not used to access an object.

CX Otherwise, it shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *malloc*( ) function shall fail if:

CX [ENOMEM] Insufficient storage space is available.

The *malloc*( ) function may fail if:

CX [EINVAL] *size* is 0 and the implementation does not support 0 sized allocations.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Some implementations set *errno* to [EAGAIN] to signal memory allocation failures that might succeed if retried and [ENOMEM] for failures that are unlikely to ever succeed, for example due to configured limits. Section 2.3 (on page 507) permits this behavior; when multiple error conditions are simultaneously true there is no precedence between them.

FUTURE DIRECTIONS

None.

SEE ALSO

aligned_alloc( ), *calloc*( ), *free*( ), *getrlimit*( ), *posix_memalign*( ), *realloc*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, the requirement to set *errno* to indicate an error is added.
- The [ENOMEM] error condition is added.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0203 [526] is applied.

Issue 8

Austin Group Defect 374 is applied, changing the RETURN VALUE and ERRORS sections in relation to 0 sized allocations.

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018 standard.

Austin Group Defects 1387 and 1489 are applied, changing the RATIONALE section.

NAME

mblen — get number of bytes in a character

SYNOPSIS

```
#include <stdlib.h>

int mblen(const char *s, size_t n);
```

DESCRIPTION

CX Except for requirements relating to data races, the functionality described on this reference page is aligned with the ISO C standard. Any other conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard for all *mblen*( ) functionality except in relation to data races.

If *s* is not a null pointer, *mblen*( ) shall determine the number of bytes constituting the character pointed to by *s*. Except that the shift state of *mbtowc*( ) is not affected, it shall be equivalent to:

```
mbtowc((wchar_t *)0, (const char *)0, 0);
mbtowc((wchar_t *)0, s, n);
```

The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls *mblen*( ).

The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a state-dependent encoding, this function shall be placed into its initial state at program startup and can be returned to that state by a call for which its character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a null pointer shall cause this function to return a non-zero value if encodings have state dependency, and 0 otherwise. If the implementation employs special bytes to change the shift state, these bytes shall not produce separate wide-character codes, but shall be grouped with an adjacent character. Changing the *LC_CTYPE* category causes the shift state of this function to be unspecified.

CX The *mblen*( ) function need not be thread-safe; however, it shall avoid data races with all other functions.

RETURN VALUE

If *s* is a null pointer, *mblen*( ) shall return a non-zero or 0 value, if character encodings, respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mblen*( ) shall either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the character (if the next *n* or fewer bytes form a valid character), or return −1 (if they do not form a CX valid character) and may set *errno* to indicate the error. In no case shall the value returned be greater than *n* or the value of the {MB_CUR_MAX} macro.

ERRORS

The *mblen*( ) function may fail if:

CX [EILSEQ] An invalid character sequence is detected. In the POSIX locale an [EILSEQ] error cannot occur since all byte values are valid characters.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

When the ISO C standard introduced threads in C11, it required *mblen*( ) to avoid data races (with itself as well as with other functions), whereas POSIX.1-2008 did not require it to be thread-safe, and in many implementations it did not avoid data races with itself and still does not. The ISO C committee intend to change the requirements in a future version of the ISO C standard, but since POSIX.1 currently refers to C17 it is necessary for it not to defer to the ISO C standard regarding data races in order to continue to allow this function not to avoid data races with itself.

FUTURE DIRECTIONS

It is expected that a change in a future version of the ISO C standard will allow a future version of this standard to remove the data race exception from the statement that it defers to the ISO C standard.

SEE ALSO

mbtowc( ), *mbstowcs*( ), *wctomb*( ), *wcstombs*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4. Aligned with the ISO C standard.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0367 [109] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0204 [663,674] is applied.

Issue 8

Austin Group Defects 708 and 1302 are applied, aligning this function with the ISO/IEC 9899:2018 standard, except in relation to data races.

NAME

mbrlen — get number of bytes in a character (restartable)

SYNOPSIS

```
#include <wchar.h>

size_t mbrlen(const char *restrict s, size_t n,
    mbstate_t *restrict ps);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

If *s* is not a null pointer, *mbrlen*( ) shall determine the number of bytes constituting the character pointed to by *s*. It shall be equivalent to:

```
mbstate_t internal;
mbrtowc(NULL, s, n, ps != NULL ? ps : &internal);
```

If *ps* is a null pointer, the *mbrlen*( ) function shall use its own internal **mbstate_t** object, which is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence. The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls *mbrlen*( ).

The behavior of this function is affected by the *LC_CTYPE* category of the current locale.

If called with a null *ps* argument, the *mbrlen*( ) function need not be thread-safe; however, such calls shall avoid data races with calls to *mbrlen*( ) with a non-null argument and with calls to all other functions.

The *mbrlen*( ) function shall not change the setting of *errno* if successful.

RETURN VALUE

The *mbrlen*( ) function shall return the first of the following that applies:

0	If the next *n* or fewer bytes complete the character that corresponds to the null wide character.
positive	If the next *n* or fewer bytes complete a valid character; the value returned shall be the number of bytes that complete the character.
(**size_t**)–2	If the next *n* bytes contribute to an incomplete but potentially valid character, and all *n* bytes have been processed. When *n* has at least the value of the {MB_CUR_MAX} macro, this case can only occur if *s* points at a sequence of redundant shift sequences (for implementations with state-dependent encodings).
(**size_t**)–1	If an encoding error occurs, in which case the next *n* or fewer bytes do not contribute to a complete and valid character. In this case, [EILSEQ] shall be stored in *errno* and the conversion state is undefined.

ERRORS

The *mbrlen*( ) function shall fail if:

CX [EILSEQ]	An invalid character sequence is detected. In the POSIX locale an [EILSEQ] error cannot occur since all byte values are valid characters.

The *mbrlen*( ) function may fail if:

[EINVAL] *ps* points to an object that contains an invalid conversion state.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mbsinit( ), *mbrtowc*( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The *mbrlen*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #170 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0368 [109,105] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0204 [663,674] is applied.

Issue 8

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018 standard.

NAME

mbrtoc16, mbrtoc32 — convert a character to a Unicode character code (restartable)

SYNOPSIS

```
#include <uchar.h>

size_t mbrtoc16(char16_t *restrict pc16, const char *restrict s,
    size_t n, mbstate_t *restrict ps);
size_t mbrtoc32(char32_t *restrict pc32, const char *restrict s,
    size_t n, mbstate_t *restrict ps);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

If *s* is a null pointer, the *mbrtoc16*( ) function shall be equivalent to the call:

```
mbrtoc16(NULL, "", 1, ps)
```

In this case, the values of the parameters *pc16* and *n* are ignored.

If *s* is not a null pointer, the *mbrtoc16*( ) function shall inspect at most *n* bytes beginning with the byte pointed to by *s* to determine the number of bytes needed to complete the next character (including any shift sequences). If the function determines that the next character is complete and valid, it shall determine the values of the corresponding wide characters and then, if *pc16* is not a null pointer, shall store the value of the first (or only) such character in the object pointed to by *pc16*. Subsequent calls shall store successive wide characters without consuming any additional input until all the characters have been stored. If the corresponding wide character is the null wide character, the resulting state described shall be the initial conversion state.

If *ps* is a null pointer, the *mbrtoc16*( ) function shall use its own internal **mbstate_t** object, which shall be initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence.

The behavior of this function is affected by the LC_CTYPE category of the current locale.

The *mbrtoc16*( ) function shall not change the setting of *errno* if successful.

The *mbrtoc32*( ) function shall behave the same way as *mbrtoc16*( ) except that the first parameter shall point to an object of type **char32_t** instead of **char16_t**. References to *pc16* in the above description shall apply as if they were *pc32* when they are being read as describing *mbrtoc32*( ).

If called with a null *ps* argument, the *mbrtoc16*( ) function need not be thread-safe; however, such calls shall avoid data races with calls to *mbrtoc16*( ) with a non-null argument and with calls to all other functions.

If called with a null *ps* argument, the *mbrtoc32*( ) function need not be thread-safe; however, such calls shall avoid data races with calls to *mbrtoc32*( ) with a non-null argument and with calls to all other functions.

The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls *mbrtoc16*( ) or *mbrtoc32*( ) with a null pointer for *ps*.

RETURN VALUE

These functions shall return the first of the following that applies:

46716 0 If the next *n* or fewer bytes complete the character that corresponds to the null
46717 wide character (which is the value stored).

46718 between 1 and *n* inclusive
46719 If the next *n* or fewer bytes complete a valid character (which is the value
46720 stored); the value returned shall be the number of bytes that complete the
46721 character.

46722 (**size_t**)–3 If the next character resulting from a previous call has been stored, in which
46723 case no bytes from the input shall be consumed by the call.

46724 (**size_t**)–2 If the next *n* bytes contribute to an incomplete but potentially valid character,
46725 and all *n* bytes have been processed (no value is stored). When *n* has at least
46726 the value of the {MB_CUR_MAX} macro, this case can only occur if *s* points at
46727 a sequence of redundant shift sequences (for implementations with state-
46728 dependent encodings).

46729 (**size_t**)–1 If an encoding error occurs, in which case the next *n* or fewer bytes do not
46730 contribute to a complete and valid character (no value is stored). In this case,
46731 [EILSEQ] shall be stored in *errno* and the conversion state is undefined.

46732 **ERRORS**

46733 These functions shall fail if:

46734 CX [EILSEQ] An invalid character sequence is detected. In the POSIX locale an [EILSEQ]
46735 error cannot occur since all byte values are valid characters.

46736 These functions may fail if:

46737 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

46738 **EXAMPLES**

46739 None.

46740 **APPLICATION USAGE**

46741 None.

46742 **RATIONALE**

46743 None.

46744 **FUTURE DIRECTIONS**

46745 None.

46746 **SEE ALSO**

46747 *c16rtomb( )*

46748 XBD **<uchar.h>**

46749 **CHANGE HISTORY**

46750 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

NAME

mbrtowc — convert a character to a wide-character code (restartable)

SYNOPSIS

```
#include <wchar.h>

size_t mbrtowc(wchar_t *restrict pwc, const char *restrict s,
    size_t n, mbstate_t *restrict ps);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

If *s* is a null pointer, the *mbrtowc*( ) function shall be equivalent to the call:

```
mbrtowc(NULL, "", 1, ps)
```

In this case, the values of the arguments *pwc* and *n* are ignored.

If *s* is not a null pointer, the *mbrtowc*( ) function shall inspect at most *n* bytes beginning at the byte pointed to by *s* to determine the number of bytes needed to complete the next character (including any shift sequences). If the function determines that the next character is completed, it shall determine the value of the corresponding wide character and then, if *pwc* is not a null pointer, shall store that value in the object pointed to by *pwc*. If the corresponding wide character is the null wide character, the resulting state described shall be the initial conversion state.

If *ps* is a null pointer, the *mbrtowc*( ) function shall use its own internal **mbstate_t** object, which shall be initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence. The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls *mbrtowc*( ).

The behavior of this function is affected by the *LC_CTYPE* category of the current locale.

If called with a null *ps* argument, the *mbrtowc*( ) function need not be thread-safe; however, such calls shall avoid data races with calls to *mbrtowc*( ) with a non-null argument and with calls to all other functions.

The *mbrtowc*( ) function shall not change the setting of *errno* if successful.

RETURN VALUE

The *mbrtowc*( ) function shall return the first of the following that applies:

0 If the next *n* or fewer bytes complete the character that corresponds to the null wide character (which is the value stored).

between 1 and *n* inclusive
: If the next *n* or fewer bytes complete a valid character (which is the value stored); the value returned shall be the number of bytes that complete the character.

(**size_t**)−2 If the next *n* bytes contribute to an incomplete but potentially valid character, and all *n* bytes have been processed (no value is stored). When *n* has at least the value of the {MB_CUR_MAX} macro, this case can only occur if *s* points at a sequence of redundant shift sequences (for implementations with state-dependent encodings).

(**size_t**)−1 If an encoding error occurs, in which case the next *n* or fewer bytes do not contribute to a complete and valid character (no value is stored). In this case, [EILSEQ] shall be stored in *errno* and the conversion state is undefined.

ERRORS

The *mbrtowc*( ) function shall fail if:

CX [EILSEQ] An invalid character sequence is detected. In the POSIX locale an [EILSEQ] error cannot occur since all byte values are valid characters.

The *mbrtowc*( ) function may fail if:

CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mbsinit( ), *mbsrtowcs*( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899: 1990/Amendment 1: 1995 (E).

Issue 6

The *mbrtowc*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EINVAL] error condition is added.

ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

Issue 7

Austin Group Interpretation 1003.1-2001 #170 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0369 [109,105] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0204 [663,674] is applied.

Issue 8

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899: 2018 standard.

NAME

mbsinit — determine conversion object status

SYNOPSIS

```
#include <wchar.h>

int mbsinit(const mbstate_t *ps);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

If *ps* is not a null pointer, the *mbsinit*( ) function shall determine whether the object pointed to by *ps* describes an initial conversion state.

RETURN VALUE

The *mbsinit*( ) function shall return non-zero if *ps* is a null pointer, or if the pointed-to object describes an initial conversion state; otherwise, it shall return zero.

If an **mbstate_t** object is altered by any of the functions described as ``restartable'', and is then used with a different character sequence, or in the other conversion direction, or with a different *LC_CTYPE* category setting than on earlier function calls, the behavior is undefined.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The **mbstate_t** object is used to describe the current conversion state from a particular character sequence to a wide-character sequence (or *vice versa*) under the rules of a particular setting of the *LC_CTYPE* category of the current locale.

The initial conversion state corresponds, for a conversion in either direction, to the beginning of a new character sequence in the initial shift state. A zero valued **mbstate_t** object is at least one way to describe an initial conversion state. A zero valued **mbstate_t** object can be used to initiate conversion involving any character sequence, in any *LC_CTYPE* category setting.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mbrlen( ), *mbrtowc*( ), *mbsrtowcs*( ), *wcrtomb*( ), *wcsrtombs*( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

NAME

mbsnrtowcs, mbsrtowcs — convert a character string to a wide-character string (restartable)

SYNOPSIS

```
#include <wchar.h>

CX  size_t mbsnrtowcs(wchar_t *restrict dst, const char **restrict src,
        size_t nmc, size_t len, mbstate_t *restrict ps);
    size_t mbsrtowcs(wchar_t *restrict dst, const char **restrict src,
        size_t len, mbstate_t *restrict ps);
```

DESCRIPTION

CX For *mbsrtowcs*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *mbsrtowcs*( ) function shall convert a sequence of characters, beginning in the conversion state described by the object pointed to by *ps*, from the array indirectly pointed to by *src* into a sequence of corresponding wide characters. If *dst* is not a null pointer, the converted characters shall be stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null character, which shall also be stored. Conversion shall stop early in either of the following cases:

- A sequence of bytes is encountered that does not form a valid character.
- *len* codes have been stored into the array pointed to by *dst* (and *dst* is not a null pointer).

Each conversion shall take place as if by a call to the *mbrtowc*( ) function.

If *dst* is not a null pointer, the pointer object pointed to by *src* shall be assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last character converted (if any). If conversion stopped due to reaching a terminating null character, and if *dst* is not a null pointer, the resulting state described shall be the initial conversion state.

If *ps* is a null pointer, the *mbsrtowcs*( ) function shall use its own internal **mbstate_t** object, which is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence.

CX The *mbsnrtowcs*( ) function shall be equivalent to the *mbsrtowcs*( ) function, except that the conversion of characters indirectly pointed to by *src* is limited to at most *nmc* bytes (the size of the input buffer), and under conditions where *mbsrtowcs*( ) would assign the address just past the last character converted (if any) to the pointer object pointed to by *src*, *mbsnrtowcs*( ) shall instead assign the address just past the last byte processed (if any) to that pointer object. If the input buffer ends with an incomplete character, conversion shall stop at the end of the input buffer; a subsequent call to *mbsnrtowcs*( ) with an input buffer that starts with the remainder of the incomplete character shall correctly complete the conversion of that character.

The behavior of these functions shall be affected by the *LC_CTYPE* category of the current locale.

The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls these functions.

CX If called with a null *ps* argument, the *mbsnrtowcs*( ) function need not be thread-safe; however, such calls shall avoid data races with calls to *mbsnrtowcs*( ) with a non-null argument and with calls to all other functions.

If called with a null *ps* argument, the *mbsrtowcs*( ) function need not be thread-safe; however,

such calls shall avoid data races with calls to *mbsrtowcs*( ) with a non-null argument and with calls to all other functions.

The *mbsrtowcs*( ) function shall not change the setting of *errno* if successful.

RETURN VALUE

If the input conversion encounters a sequence of bytes that do not form a valid character, an encoding error occurs. In this case, these functions shall store the value of the macro [EILSEQ] in *errno* and shall return (**size_t**)–1; the conversion state is undefined. Otherwise, these functions shall return the number of characters successfully converted, not including the terminating null (if any).

ERRORS

These functions shall fail if:

CX [EILSEQ] An invalid character sequence is detected. In the POSIX locale an [EILSEQ] error cannot occur since all byte values are valid characters.

These functions may fail if:

CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iconv( ), *mbrtowc*( ), *mbsinit*( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The *mbsrtowcs*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

The [EINVAL] error condition is marked CX.

Issue 7

Austin Group Interpretation 1003.1-2001 #170 is applied.

The *mbsnrtowcs*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0370 [109,105] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0205 [601], XSH/TC2-2008/0206 [663], and XSH/TC2-2008/0207 [601] are applied.

Issue 8

Austin Group Defect 616 is applied, requiring that when the *mbsnrtowcs*( ) input buffer ends with an incomplete character, conversion stops at the end of the input buffer (not at the end of the previous character, if any).

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

NAME

mbstowcs — convert a character string to a wide-character string

SYNOPSIS

```
#include <stdlib.h>

size_t mbstowcs(wchar_t *restrict pwcs, const char *restrict s,
    size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *mbstowcs*( ) function shall convert a sequence of characters that begins in the initial shift state from the array pointed to by *s* into a sequence of corresponding wide-character codes and shall store not more than *n* wide-character codes into the array pointed to by *pwcs*. No characters that follow a null byte (which is converted into a wide-character code with value 0) shall be examined or converted. Each character shall be converted as if by a call to *mbtowc*( ), except that the shift state of *mbtowc*( ) is not affected.

No more than *n* elements shall be modified in the array pointed to by *pwcs*. If copying takes place between objects that overlap, the behavior is undefined.

XSI The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale. If *pwcs* is a null pointer, *mbstowcs*( ) shall return the length required to convert the entire array regardless of the value of *n*, but no values are stored.

RETURN VALUE

CX If an invalid character is encountered, *mbstowcs*( ) shall return (**size_t**)–1 and shall set *errno* to indicate the error.

XSI Otherwise, *mbstowcs*( ) shall return the number of the array elements modified (or required if *pwcs* is null), not including a terminating 0 code, if any. The array shall not be zero-terminated if the value returned is *n*.

ERRORS

The *mbstowcs*( ) function shall fail if:

CX [EILSEQ] An invalid character sequence is detected. In the POSIX locale an [EILSEQ] error cannot occur since all byte values are valid characters.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mblen( ), *mbtowc*( ), *wctomb*( ), *wcstombs*( )

XBD **<stdlib.h>**

47001 **CHANGE HISTORY**

47002 First released in Issue 4. Aligned with the ISO C standard.

47003 **Issue 6**

47004 The *mbstowcs*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

47005 Extensions beyond the ISO C standard are marked.

47006 **Issue 7**

47007 Austin Group Interpretation 1003.1-2001 #170 is applied.

47008 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0371 [195] is applied.

47009 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0208 [663,674] is applied.

47010 **NAME**

47011 mbtowc — convert a character to a wide-character code

47012 **SYNOPSIS**

47013 ```
#include <stdlib.h>
```

47014 ```
int mbtowc(wchar_t *restrict pwc, const char *restrict s, size_t n);
```

47015 **DESCRIPTION**

47016 CX Except for requirements relating to data races, the functionality described on this reference page
47017 is aligned with the ISO C standard. Any other conflict between the requirements described here
47018 and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C
47019 standard for all *mbtowc*( ) functionality except in relation to data races.

47020 If *s* is not a null pointer, *mbtowc*( ) shall determine the number of bytes that constitute the
47021 character pointed to by *s*. It shall then determine the wide-character code for the value of type
47022 **wchar_t** that corresponds to that character. (The value of the wide-character code corresponding
47023 to the null byte is 0.) If the character is valid and *pwc* is not a null pointer, *mbtowc*( ) shall store
47024 the wide-character code in the object pointed to by *pwc*.

47025 The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a
47026 state-dependent encoding, this function shall be placed into its initial state at program startup
47027 and can be returned to that state by a call for which its character pointer argument, *s*, is a null
47028 pointer. Subsequent calls with *s* as other than a null pointer shall cause the internal state of the
47029 function to be altered as necessary. A call with *s* as a null pointer shall cause this function to
47030 return a non-zero value if encodings have state dependency, and 0 otherwise. If the
47031 implementation employs special bytes to change the shift state, these bytes shall not produce
47032 separate wide-character codes, but shall be grouped with an adjacent character. Changing the
47033 *LC_CTYPE* category causes the shift state of this function to be unspecified. At most *n* bytes of
47034 the array pointed to by *s* shall be examined.

47035 The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls
47036 *mbtowc*( ).

47037 CX The *mbtowc*( ) function need not be thread-safe; however, it shall avoid data races with all other
47038 functions.

47039 **RETURN VALUE**

47040 If *s* is a null pointer, *mbtowc*( ) shall return a non-zero or 0 value, if character encodings,
47041 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mbtowc*( )
47042 shall either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the
47043 CX converted character (if the next *n* or fewer bytes form a valid character), or return –1 and shall
47044 set *errno* to indicate the error (if they do not form a valid character).

47045 In no case shall the value returned be greater than *n* or the value of the {MB_CUR_MAX} macro.

47046 **ERRORS**

47047 The *mbtowc*( ) function shall fail if:

47048 CX [EILSEQ] An invalid character sequence is detected. In the POSIX locale an [EILSEQ]
47049 error cannot occur since all byte values are valid characters.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

When the ISO C standard introduced threads in C11, it required *mbtowc*( ) to avoid data races (with itself as well as with other functions), whereas POSIX.1-2008 did not require it to be thread-safe, and in many implementations it did not avoid data races with itself and still does not. The ISO C committee intend to change the requirements in a future version of the ISO C standard, but since POSIX.1 currently refers to C17 it is necessary for it not to defer to the ISO C standard regarding data races in order to continue to allow this function not to avoid data races with itself.

FUTURE DIRECTIONS

It is expected that a change in a future version of the ISO C standard will allow a future version of this standard to remove the data race exception from the statement that it defers to the ISO C standard.

SEE ALSO

mblen( ), *mbstowcs*( ), *wctomb*( ), *wcstombs*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4. Aligned with the ISO C standard.

Issue 6

The *mbtowc*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

Extensions beyond the ISO C standard are marked.

Issue 7

Austin Group Interpretation 1003.1-2001 #170 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0372 [109] and XSH/TC1-2008/0373 [195] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0209 [663,674] is applied.

Issue 8

Austin Group Defects 708 and 1302 are applied, aligning this function with the ISO/IEC 9899: 2018 standard, except in relation to data races.

47082 **NAME**

47083 memccpy — copy bytes in memory

47084 **SYNOPSIS**

47085 XSI
```
#include <string.h>

void *memccpy(void *restrict s1, const void *restrict s2,
    int c, size_t n);
```

47088 **DESCRIPTION**

47089 The *memccpy*( ) function shall copy bytes from memory area *s2* into *s1*, stopping after the first
47090 occurrence of byte *c* (converted to an **unsigned char**) is copied, or after *n* bytes are copied,
47091 whichever comes first. If copying takes place between objects that overlap, the behavior is
47092 undefined.

47093 The *memccpy*( ) function shall not change the setting of *errno* on valid input.

47094 **RETURN VALUE**

47095 The *memccpy*( ) function shall return a pointer to the byte after the copy of *c* in *s1*, or a null
47096 pointer if *c* was not found in the first *n* bytes of *s2*.

47097 **ERRORS**

47098 No errors are defined.

47099 **EXAMPLES**

47100 None.

47101 **APPLICATION USAGE**

47102 The *memccpy*( ) function does not check for the overflow of the receiving memory area.

47103 **RATIONALE**

47104 None.

47105 **FUTURE DIRECTIONS**

47106 None.

47107 **SEE ALSO**

47108 XBD **<string.h>**

47109 **CHANGE HISTORY**

47110 First released in Issue 1. Derived from Issue 1 of the SVID.

47111 **Issue 6**

47112 The **restrict** keyword is added to the *memccpy*( ) prototype for alignment with the
47113 ISO/IEC 9899:1999 standard.

47114 **Issue 8**

47115 Austin Group Defect 448 is applied, adding a requirement that *memccpy*( ) does not change the
47116 setting of *errno* on valid input.

47117 **NAME**

47118 memchr — find byte in memory

47119 **SYNOPSIS**

47120 ```
#include <string.h>
```

47121 ```
void *memchr(const void *s, int c, size_t n);
```

47122 **DESCRIPTION**

47123 CX The functionality described on this reference page is aligned with the ISO C standard. Any
47124 conflict between the requirements described here and the ISO C standard is unintentional. This
47125 volume of POSIX.1-2024 defers to the ISO C standard.

47126 The *memchr*( ) function shall locate the first occurrence of *c* (converted to an **unsigned char**) in
47127 the initial *n* bytes (each interpreted as **unsigned char**) pointed to by *s*.

47128 The implementation shall behave as if it reads the bytes sequentially and stops as soon as a
47129 matching byte is found.

47130 CX The *memchr*( ) function shall not change the setting of *errno* on valid input.

47131 **RETURN VALUE**

47132 The *memchr*( ) function shall return a pointer to the located byte, or a null pointer if the byte is
47133 not found.

47134 **ERRORS**

47135 No errors are defined.

47136 **EXAMPLES**

47137 None.

47138 **APPLICATION USAGE**

47139 None.

47140 **RATIONALE**

47141 None.

47142 **FUTURE DIRECTIONS**

47143 None.

47144 **SEE ALSO**

47145 XBD **<string.h>**

47146 **CHANGE HISTORY**

47147 First released in Issue 1. Derived from Issue 1 of the SVID.

47148 **Issue 7**

47149 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0374 [110] is applied.

47150 **Issue 8**

47151 Austin Group Defect 448 is applied, adding a requirement that *memchr*( ) does not change the
47152 setting of *errno* on valid input.

47153 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018
47154 standard.

NAME

memcmp — compare bytes in memory

SYNOPSIS

```
#include <string.h>

int memcmp(const void *s1, const void *s2, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *memcmp*( ) function shall compare the first *n* bytes (each interpreted as **unsigned char**) of the object pointed to by *s1* to the first *n* bytes of the object pointed to by *s2*.

The sign of a non-zero return value shall be determined by the sign of the difference between the values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the objects being compared.

CX The *memcmp*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

The *memcmp*( ) function shall return an integer greater than, equal to, or less than 0, if the object pointed to by *s1* is greater than, equal to, or less than the object pointed to by *s2*, respectively.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *memcmp*( ) does not change the setting of *errno* on valid input.

47190 **NAME**

47191 memcpy — copy bytes in memory

47192 **SYNOPSIS**

47193 ```
#include <string.h>
```

47194 ```
void *memcpy(void *restrict s1, const void *restrict s2, size_t n);
```

47195 **DESCRIPTION**

47196 CX The functionality described on this reference page is aligned with the ISO C standard. Any
47197 conflict between the requirements described here and the ISO C standard is unintentional. This
47198 volume of POSIX.1-2024 defers to the ISO C standard.

47199 The *memcpy*( ) function shall copy *n* bytes from the object pointed to by *s2* into the object pointed
47200 to by *s1*. If copying takes place between objects that overlap, the behavior is undefined.

47201 CX The *memcpy*( ) function shall not change the setting of *errno* on valid input.

47202 **RETURN VALUE**

47203 The *memcpy*( ) function shall return *s1*; no return value is reserved to indicate an error.

47204 **ERRORS**

47205 No errors are defined.

47206 **EXAMPLES**

47207 None.

47208 **APPLICATION USAGE**

47209 The *memcpy*( ) function does not check for the overflow of the receiving memory area.

47210 **RATIONALE**

47211 None.

47212 **FUTURE DIRECTIONS**

47213 None.

47214 **SEE ALSO**

47215 XBD **<string.h>**

47216 **CHANGE HISTORY**

47217 First released in Issue 1. Derived from Issue 1 of the SVID.

47218 **Issue 6**

47219 The *memcpy*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

47220 **Issue 8**

47221 Austin Group Defect 448 is applied, adding a requirement that *memcpy*( ) does not change the
47222 setting of *errno* on valid input.

47223 **NAME**

47224 memmem — find a byte subsequence in a byte sequence

47225 **SYNOPSIS**

47226 CX
```
#include <string.h>

void *memmem(const void *haystack, size_t haystacklen,
             const void *needle, size_t needlelen);
```

47229 **DESCRIPTION**

47230 The *memmem*( ) function shall locate the first occurrence of byte sequence *needle* of length
47231 *needlelen* in byte sequence *haystack* of length *haystacklen*.

47232 **RETURN VALUE**

47233 Upon successful completion, *memmem*( ) shall return a pointer to the the first byte of the located
47234 byte sequence in *haystack*, or a null pointer if the byte sequence is not found.

47235 If *needlelen* is zero, the function shall return *haystack*.

47236 If *haystacklen* is less than *needlelen*, the function shall return a null pointer.

47237 **ERRORS**

47238 No errors are defined.

47239 **EXAMPLES**

47240 None.

47241 **APPLICATION USAGE**

47242 None.

47243 **RATIONALE**

47244 This function is similar to *strstr*( ), except that NUL bytes may be included in either *needle* or
47245 *haystack*.

47246 **FUTURE DIRECTIONS**

47247 None.

47248 **SEE ALSO**

47249 *memchr*( ), *strstr*( )

47250 XBD **<string.h>**

47251 **CHANGE HISTORY**

47252 First released in Issue 8.

47253 **NAME**

47254 memmove — copy bytes in memory with overlapping areas

47255 **SYNOPSIS**

47256 ```
#include <string.h>
```

47257 ```
void *memmove(void *s1, const void *s2, size_t n);
```

47258 **DESCRIPTION**

47259 CX The functionality described on this reference page is aligned with the ISO C standard. Any
47260 conflict between the requirements described here and the ISO C standard is unintentional. This
47261 volume of POSIX.1-2024 defers to the ISO C standard.

47262 The *memmove*( ) function shall copy *n* bytes from the object pointed to by *s2* into the object
47263 pointed to by *s1*. Copying takes place as if the *n* bytes from the object pointed to by *s2* are first
47264 copied into a temporary array of *n* bytes that does not overlap the objects pointed to by *s1* and
47265 *s2*, and then the *n* bytes from the temporary array are copied into the object pointed to by *s1*.

47266 CX The *memmove*( ) function shall not change the setting of *errno* on valid input.

47267 **RETURN VALUE**

47268 The *memmove*( ) function shall return *s1*; no return value is reserved to indicate an error.

47269 **ERRORS**

47270 No errors are defined.

47271 **EXAMPLES**

47272 None.

47273 **APPLICATION USAGE**

47274 None.

47275 **RATIONALE**

47276 None.

47277 **FUTURE DIRECTIONS**

47278 None.

47279 **SEE ALSO**

47280 XBD **<string.h>**

47281 **CHANGE HISTORY**

47282 First released in Issue 4. Derived from the ANSI C standard.

47283 **Issue 8**

47284 Austin Group Defect 448 is applied, adding a requirement that *memmove*( ) does not change the
47285 setting of *errno* on valid input.

47286 **NAME**

47287 memset — set bytes in memory

47288 **SYNOPSIS**

47289
```
#include <string.h>
```

47290
```
void *memset(void *s, int c, size_t n);
```

47291 **DESCRIPTION**

47292 CX The functionality described on this reference page is aligned with the ISO C standard. Any
47293 conflict between the requirements described here and the ISO C standard is unintentional. This
47294 volume of POSIX.1-2024 defers to the ISO C standard.

47295 The *memset*( ) function shall copy *c* (converted to an **unsigned char**) into each of the first *n* bytes
47296 of the object pointed to by *s*.

47297 CX The *memset*( ) function shall not change the setting of *errno* on valid input.

47298 **RETURN VALUE**

47299 The *memset*( ) function shall return *s*; no return value is reserved to indicate an error.

47300 **ERRORS**

47301 No errors are defined.

47302 **EXAMPLES**

47303 None.

47304 **APPLICATION USAGE**

47305 None.

47306 **RATIONALE**

47307 None.

47308 **FUTURE DIRECTIONS**

47309 None.

47310 **SEE ALSO**

47311 XBD **<string.h>**

47312 **CHANGE HISTORY**

47313 First released in Issue 1. Derived from Issue 1 of the SVID.

47314 **Issue 8**

47315 Austin Group Defect 448 is applied, adding a requirement that *memset*( ) does not change the
47316 setting of *errno* on valid input.

47317 **NAME**

47318 mkdir, mkdirat — make a directory

47319 **SYNOPSIS**

47320 ```
#include <sys/stat.h>
```

47321 ```
int mkdir(const char *path, mode_t mode);
```

47322 OH ```
#include <fcntl.h>
```

47323 ```
int mkdirat(int fd, const char *path, mode_t mode);
```

47324 **DESCRIPTION**

47325 XSI The *mkdir*( ) function shall create a new directory with name *path*. The file permission bits and
47326 S_ISVTX bit of the new directory shall be initialized from *mode*. The file permission bits of the
47327 *mode* argument shall be modified by the file creation mask of the process.

47328 XSI When bits in *mode* other than the file permission bits and S_ISVTX are set, the meaning of these
47329 additional bits is implementation-defined.

47330 The directory's user ID shall be set to the process' effective user ID. The directory's group ID
47331 shall be set to the group ID of the parent directory or to the effective group ID of the process.
47332 Implementations shall provide a way to initialize the directory's group ID to the group ID of the
47333 parent directory. Implementations may, but need not, provide an implementation-defined way
47334 to initialize the directory's group ID to the effective group ID of the calling process.

47335 The newly created directory shall be an empty directory.

47336 If *path* names a symbolic link, *mkdir*( ) shall fail and set *errno* to [EEXIST].

47337 Upon successful completion, *mkdir*( ) shall mark for update the last data access, last data
47338 modification, and last file status change timestamps of the directory. Also, the last data
47339 modification and last file status change timestamps of the directory that contains the new entry
47340 shall be marked for update.

47341 The *mkdirat*( ) function shall be equivalent to the *mkdir*( ) function except in the case where *path*
47342 specifies a relative path. In this case the newly created directory is created relative to the
47343 directory associated with the file descriptor *fd* instead of the current working directory. If the
47344 access mode of the open file description associated with the file descriptor is not O_SEARCH,
47345 the function shall check whether directory searches are permitted using the current permissions
47346 of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function
47347 shall not perform the check.

47348 If *mkdirat*( ) is passed the special value AT_FDCWD in the *fd* parameter, the current working
47349 directory shall be used and the behavior shall be identical to a call to *mkdir*( ).

47350 **RETURN VALUE**

47351 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
47352 return –1 and set *errno* to indicate the error. If –1 is returned, no directory shall be created.

47353 **ERRORS**

47354 These functions shall fail if:

47355 [EACCES] Search permission is denied on a component of the path prefix, or write
47356 permission is denied on the parent directory of the directory to be created.

47357 [EEXIST] The named file exists.

47358 [EILSEQ] The last pathname component of *path* is not a portable filename, and cannot be
47359 created in the target directory.

47360 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
47361 argument.

47362 [EMLINK] The link count of the parent directory would exceed {LINK_MAX}.

47363 [ENAMETOOLONG]
47364 The length of a component of a pathname is longer than {NAME_MAX}.

47365 [ENOENT] A component of the path prefix of *path* does not name an existing file or *path* is
47366 an empty string.

47367 [ENOSPC] The file system does not contain enough space to hold the contents of the new
47368 directory or to extend the parent directory of the new directory.

47369 [ENOTDIR] A component of the path prefix names an existing file that is neither a
47370 directory nor a symbolic link to a directory.

47371 [EROFS] The parent directory resides on a read-only file system.

47372 In addition, the *mkdirat*( ) function shall fail if:

47373 [EACCES] The access mode of the open file description associated with *fd* is not
47374 O_SEARCH and the permissions of the directory underlying *fd* do not permit
47375 directory searches.

47376 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
47377 neither AT_FDCWD nor a valid file descriptor open for reading or searching.

47378 [ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated
47379 with a non-directory file.

47380 These functions may fail if:

47381 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
47382 resolution of the *path* argument.

47383 [ENAMETOOLONG]
47384 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
47385 symbolic link produced an intermediate result with a length that exceeds
47386 {PATH_MAX}.

47387 **EXAMPLES**

47388 **Creating a Directory**

47389 The following example shows how to create a directory named **/home/cnd/mod1**, with
47390 read/write/search permissions for owner and group, and with read/search permissions for
47391 others.

```
47392 #include <sys/types.h>
47393 #include <sys/stat.h>

47394 int status;
47395 ...
47396 status = mkdir("/home/cnd/mod1", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
```

47397 **APPLICATION USAGE**

47398 None.

RATIONALE

The *mkdir*( ) function originated in 4.2 BSD and was added to System V in Release 3.0.

4.3 BSD detects [ENAMETOOLONG].

The POSIX.1-1990 standard required that the group ID of a newly created directory be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required that implementations provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also supporting a way to set the group ID to the effective group ID of the creating process. Conforming applications should not assume which group ID will be used. If it matters, an application can use *chown*( ) to set the group ID after the directory is created, or determine under what conditions the implementation will set the desired group ID.

The purpose of the *mkdirat*( ) function is to create a directory in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to the call to *mkdir*( ), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *mkdirat*( ) function it can be guaranteed that the newly created directory is located relative to the desired directory.

Implementations are encouraged to have *mkdir*( ) and *mkdirat*( ) report an [EILSEQ] error if the last component of *path* contains any bytes that have the encoded value of a <newline> character.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *mkdtemp*( ), *mknod*( ), *umask*( )

XBD **<fcntl.h>**, **<sys/stat.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

The *mkdirat*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0375 [461], XSH/TC1-2008/0376 [324],

XSH/TC1-2008/0377 [277], XSH/TC1-2008/0378 [278], and XSH/TC1-2008/0379 [278] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0210 [873], XSH/TC2-2008/0211 [591], XSH/TC2-2008/0212 [817], XSH/TC2-2008/0213 [817], and XSH/TC2-2008/0214 [591] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defects 293 and 1734 are applied, adding the [EILSEQ] error.

Austin Group Defect 1522 is applied, adding requirements relating to the S_ISVTX bit.

Austin Group Defect 1729 is applied, changing the description of the [ENOENT] error.

NAME

mkdtemp, mkostemp, mkstemp — create a unique directory or file

SYNOPSIS

CX
```
#include <stdlib.h>

char *mkdtemp(char *template);
int mkostemp(char *template, int flag);
int mkstemp(char *template);
```

DESCRIPTION

The *mkdtemp*( ) function shall create a directory with a unique name derived from *template*. The application shall ensure that the string provided in *template* is a pathname ending with at least six trailing 'X' characters. The *mkdtemp*( ) function shall modify the contents of *template* by replacing six or more 'X' characters at the end of the pathname with the same number of characters from the portable filename character set. The characters shall be chosen such that the resulting pathname does not duplicate the name of an existing file at the time of the call to *mkdtemp*( ). The *mkdtemp*( ) function shall use the resulting pathname to create the new directory as if by a call to:

```
mkdir(pathname, S_IRWXU)
```

The *mkstemp*( ) function shall create a regular file with a unique name derived from *template* and return a file descriptor for the file open for reading and writing. The application shall ensure that the string provided in *template* is a pathname ending with at least six trailing 'X' characters. The *mkstemp*( ) function shall modify the contents of *template* by replacing six or more 'X' characters at the end of the pathname with the same number of characters from the portable filename character set. The characters shall be chosen such that the resulting pathname does not duplicate the name of an existing file at the time of the call to *mkstemp*( ). The *mkstemp*( ) function shall use the resulting pathname to create the file, and obtain a file descriptor for it, as if by a call to:

```
open(pathname, O_RDWR|O_CREAT|O_EXCL, S_IRUSR|S_IWUSR)
```

By behaving as if the O_EXCL flag for *open*( ) is set, the function prevents any possible race condition between testing whether the file exists and opening it for use.

The *mkostemp*( ) function shall be equivalent to the *mkstemp*( ) function, except that the *flag* argument can contain additional flags to be used as if by *open*( ). Behavior is unspecified if the *flag* argument contains more than the following flags:

	O_APPEND	Set append mode.
	O_CLOEXEC	Set the FD_CLOEXEC file descriptor flag.
	O_CLOFORK	Set the FD_CLOFORK file descriptor flag.
SIO	O_DSYNC	Write according to the synchronized I/O data integrity completion.
SIO	O_RSYNC	Synchronized read I/O operations.
XSI|SIO	O_SYNC	Write according to synchronized I/O file integrity completion.

RETURN VALUE

Upon successful completion, the *mkdtemp*( ) function shall return the value of *template*. Otherwise, it shall return a null pointer and shall set *errno* to indicate the error.

Upon successful completion, the *mkstemp*( ) function shall return an open file descriptor. Otherwise, it shall return –1 and shall set *errno* to indicate the error.

ERRORS

These functions shall fail if:

[EINVAL] The string pointed to by *template* does not end in `"XXXXXX"`.

The *mkostemp*( ) function may fail if:

[EINVAL] The value of the *flag* argument is invalid.

Additional error conditions for the *mkdtemp*( ) function are defined in *mkdir*( ). Additional error conditions for the *mkstemp*( ) and *mkostemp*( ) functions are defined in *open*( ).

EXAMPLES

Generating a Pathname

The following example creates a file with a 10-character name beginning with the characters `"file"` and opens the file for reading and writing. The value returned as the value of *fd* is a file descriptor that identifies the file.

```
#include <stdlib.h>
...
char template[] = "/tmp/fileXXXXXX";
int fd;

fd = mkstemp(template);
```

APPLICATION USAGE

It is possible to run out of letters.

Portable applications should pass exactly six trailing `'X'`s in the template and no more; implementations may treat any additional trailing `'X'`s as either a fixed or replaceable part of the template. To be sure of only passing six, a fixed string of at least one non-`'X'` character should precede the six `'X'`s.

Since `'X'` is in the portable filename character set, some of the replacement characters can be `'X'`s, leaving part (or even all) of the template effectively unchanged.

RATIONALE

The O_CLOEXEC and O_CLOFORK flags of *mkostemp*( ) are necessary to avoid a data race in multi-threaded applications. Without O_CLOFORK, a file descriptor is leaked into a child process created by one thread in the window between another thread creating a temporary file descriptor with *mkstemp*( ) and then using *fcntl*( ) to set the FD_CLOFORK flag. Without O_CLOEXEC, a temporary file descriptor intentionally inherited by child processes is similarly leaked into an executed program if FD_CLOEXEC is not set atomically.

Implementations are encouraged to have *mkdtemp*( ), *mkostemp*( ), and *mkstemp*( ) report an [EILSEQ] error if the last component of the pathname in *template* contains any bytes that have the encoded value of a <newline> character.

FUTURE DIRECTIONS

None.

SEE ALSO

getpid( ), *mkdir*( ), *open*( ), *tmpfile*( ), *tmpnam*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

SD5-XSH-ERN-168 is applied, clarifying file permissions upon creation.

The *mkstemp*( ) function is moved from the XSI option to the Base.

The *mkdtemp*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0380 [291], XSH/TC1-2008/0381 [324], and XSH/TC1-2008/0382 [291] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0215 [567,669] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defects 411, 1318, and 1350 are applied, adding *mkostemp*( ).

Austin Group Defect 652 is applied, adding the [EINVAL] error for *mkstemp*( ).

Austin Group Defect 1734 is applied, replacing the error conditions specified only for *mkdtemp*( ) with a reference to *mkdir*( ).

NAME

mkfifo, mkfifoat — make a FIFO special file

SYNOPSIS

```
#include <sys/stat.h>

int mkfifo(const char *path, mode_t mode);
```

OH

```
#include <fcntl.h>

int mkfifoat(int fd, const char *path, mode_t mode);
```

DESCRIPTION

The *mkfifo*( ) function shall create a new FIFO special file named by the pathname pointed to by *path*. The file permission bits of the new FIFO shall be initialized from *mode*. The file permission bits of the *mode* argument shall be modified by the process' file creation mask.

When bits in *mode* other than the file permission bits are set, the effect is implementation-defined.

If *path* names a symbolic link, *mkfifo*( ) shall fail and set *errno* to [EEXIST].

The FIFO's user ID shall be set to the process' effective user ID. The FIFO's group ID shall be set to the group ID of the parent directory or to the effective group ID of the process. Implementations shall provide a way to initialize the FIFO's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the FIFO's group ID to the effective group ID of the calling process.

Upon successful completion, *mkfifo*( ) shall mark for update the last data access, last data modification, and last file status change timestamps of the file. Also, the last data modification and last file status change timestamps of the directory that contains the new entry shall be marked for update.

The *mkfifoat*( ) function shall be equivalent to the *mkfifo*( ) function except in the case where *path* specifies a relative path. In this case the newly created FIFO is created relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

If *mkfifoat*( ) is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used and the behavior shall be identical to a call to *mkfifo*( ).

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return –1 and set *errno* to indicate the error. If –1 is returned, no FIFO shall be created.

ERRORS

These functions shall fail if:

[EACCES] A component of the path prefix denies search permission, or write permission is denied on the parent directory of the FIFO to be created.

[EEXIST] The named file already exists.

[EILSEQ] The last pathname component of *path* is not a portable filename, and cannot be created in the target directory.

47596 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
47597 argument.

47598 [ENAMETOOLONG]
47599 The length of a component of a pathname is longer than {NAME_MAX}.

47600 [ENOENT] A component of the path prefix of *path* does not name an existing file or *path* is
47601 an empty string.

47602 [ENOENT] or [ENOTDIR]
47603 The *path* argument contains at least one non-<slash> character and ends with
47604 one or more trailing <slash> characters. If *path* without the trailing <slash>
47605 characters would name an existing file, an [ENOENT] error shall not occur.

47606 [ENOSPC] The directory that would contain the new file cannot be extended or the file
47607 system is out of file-allocation resources.

47608 [ENOTDIR] A component of the path prefix names an existing file that is neither a
47609 directory nor a symbolic link to a directory.

47610 [EROFS] The named file resides on a read-only file system.

47611 The *mkfifoat*( ) function shall fail if:

47612 [EACCES] The access mode of the open file description associated with *fd* is not
47613 O_SEARCH and the permissions of the directory underlying *fd* do not permit
47614 directory searches.

47615 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
47616 neither AT_FDCWD nor a valid file descriptor open for reading or searching.

47617 [ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated
47618 with a non-directory file.

47619 These functions may fail if:

47620 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
47621 resolution of the *path* argument.

47622 [ENAMETOOLONG]
47623 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
47624 symbolic link produced an intermediate result with a length that exceeds
47625 {PATH_MAX}.

47626 **EXAMPLES**

47627 **Creating a FIFO File**

47628 The following example shows how to create a FIFO file named **/home/cnd/mod_done**, with
47629 read/write permissions for owner, and with read permissions for group and others.

```
47630 #include <sys/types.h>
47631 #include <sys/stat.h>

47632 int status;
47633 ...
47634 status = mkfifo("/home/cnd/mod_done", S_IWUSR | S_IRUSR |
47635     S_IRGRP | S_IROTH);
```


47636 **APPLICATION USAGE**

47637 None.

47638 **RATIONALE**

47639 The syntax of this function is intended to maintain compatibility with historical
47640 implementations of *mknod*( ). The latter function was included in the 1984 /usr/group standard
47641 but only for use in creating FIFO special files. The *mknod*( ) function was originally excluded
47642 from the POSIX.1-1988 standard as implementation-defined and replaced by *mkdir*( ) and
47643 *mkfifo*( ). The *mknod*( ) function is now included for alignment with the Single UNIX
47644 Specification.

47645 The POSIX.1-1990 standard required that the group ID of a newly created FIFO be set to the
47646 group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2
47647 required that implementations provide a way to have the group ID be set to the group ID of the
47648 containing directory, but did not prohibit implementations also supporting a way to set the
47649 group ID to the effective group ID of the creating process. Conforming applications should not
47650 assume which group ID will be used. If it matters, an application can use *chown*( ) to set the
47651 group ID after the FIFO is created, or determine under what conditions the implementation will
47652 set the desired group ID.

47653 The purpose of the *mkfifoat*( ) function is to create a FIFO special file in directories other than the
47654 current working directory without exposure to race conditions. Any part of the path of a file
47655 could be changed in parallel to a call to *mkfifo*( ), resulting in unspecified behavior. By opening a
47656 file descriptor for the target directory and using the *mkfifoat*( ) function it can be guaranteed that
47657 the newly created FIFO is located relative to the desired directory.

47658 Implementations are encouraged to have *mkfifo*( ) and *mkfifoat*( ) report an [EILSEQ] error if the
47659 last component of *path* contains any bytes that have the encoded value of a <newline> character.

47660 **FUTURE DIRECTIONS**

47661 None.

47662 **SEE ALSO**

47663 *chmod*( ), *mknod*( ), *umask*( )

47664 XBD **<fcntl.h>**, **<sys/stat.h>**, **<sys/types.h>**

47665 **CHANGE HISTORY**

47666 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

47667 **Issue 6**

47668 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

47669 The following new requirements on POSIX implementations derive from alignment with the
47670 Single UNIX Specification:

47671 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
47672 required for conforming implementations of previous POSIX specifications, it was not
47673 required for UNIX applications.

47674 • The [ELOOP] mandatory error condition is added.

47675 • A second [ENAMETOOLONG] is added as an optional error condition.

47676 The following changes were made to align with the IEEE P1003.1a draft standard:

47677 • The [ELOOP] optional error condition is added.

47678 **Issue 7**

47679 Austin Group Interpretation 1003.1-2001 #143 is applied.

47680 The *mkfifoat*( ) function is added from The Open Group Technical Standard, 2006, Extended API
47681 Set Part 2.

47682 Changes are made related to support for finegrained timestamps.

47683 Changes are made to allow a directory to be opened for searching.

47684 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0383 [461], XSH/TC1-2008/0384
47685 [146,435], XSH/TC1-2008/0385 [324], XSH/TC1-2008/0386 [278], and XSH/TC1-2008/0387
47686 [278] are applied.

47687 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0216 [873], XSH/TC2-2008/0217 [591],
47688 XSH/TC2-2008/0218 [817], XSH/TC2-2008/0219 [822], XSH/TC2-2008/0220 [817], and
47689 XSH/TC2-2008/0221 [591] are applied.

47690 **Issue 8**

47691 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of
47692 filenames containing any bytes that have the encoded value of a <newline> character.

47693 Austin Group Defect 293 is applied, adding the [EILSEQ] error.

NAME

mknod, mknodat — make directory, special file, or regular file

SYNOPSIS

XSI
```
#include <sys/stat.h>

int mknod(const char *path, mode_t mode, dev_t dev);
```

OH XSI
```
#include <fcntl.h>
```

XSI
```
int mknodat(int fd, const char *path, mode_t mode, dev_t dev);
```

DESCRIPTION

The *mknod*( ) function shall create a new file named by the pathname to which the argument *path* points.

The file type for *path* is OR'ed into the *mode* argument, and the application shall select one of the following symbolic constants:

Name	Description
S_IFIFO	FIFO-special
S_IFCHR	Character-special (non-portable)
S_IFDIR	Directory (non-portable)
S_IFBLK	Block-special (non-portable)
S_IFREG	Regular (non-portable)

The only portable use of *mknod*( ) is to create a FIFO-special file. If *mode* is not S_IFIFO or *dev* is not 0, the behavior of *mknod*( ) is unspecified.

The permissions for the new file are OR'ed into the *mode* argument, and may be selected from any combination of the following symbolic constants:

Name	Description
S_ISUID	Set user ID on execution.
S_ISGID	Set group ID on execution.
S_IRWXU	Read, write, or execute (search) by owner.
S_IRUSR	Read by owner.
S_IWUSR	Write by owner.
S_IXUSR	Execute (search) by owner.
S_IRWXG	Read, write, or execute (search) by group.
S_IRGRP	Read by group.
S_IWGRP	Write by group.
S_IXGRP	Execute (search) by group.
S_IRWXO	Read, write, or execute (search) by others.
S_IROTH	Read by others.
S_IWOTH	Write by others.
S_IXOTH	Execute (search) by others.
S_ISVTX	On directories, restricted deletion flag.

The user ID of the file shall be initialized to the effective user ID of the process. The group ID of the file shall be initialized to either the effective group ID of the process or the group ID of the parent directory. Implementations shall provide a way to initialize the file's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the file's group ID to the effective group ID of the

47737 calling process. The owner, group, and other permission bits of *mode* shall be modified by the file
47738 mode creation mask of the process. The *mknod*( ) function shall clear each bit whose
47739 corresponding bit in the file mode creation mask of the process is set.

47740 If *path* names a symbolic link, *mknod*( ) shall fail and set *errno* to [EEXIST].

47741 Upon successful completion, *mknod*( ) shall mark for update the last data access, last data
47742 modification, and last file status change timestamps of the file. Also, the last data modification
47743 and last file status change timestamps of the directory that contains the new entry shall be
47744 marked for update.

47745 Only a process with appropriate privileges may invoke *mknod*( ) for file types other than FIFO-
47746 special.

47747 The *mknodat*( ) function shall be equivalent to the *mknod*( ) function except in the case where *path*
47748 specifies a relative path. In this case the newly created directory, special file, or regular file is
47749 located relative to the directory associated with the file descriptor *fd* instead of the current
47750 working directory. If the access mode of the open file description associated with the file
47751 descriptor is not O_SEARCH, the function shall check whether directory searches are permitted
47752 using the current permissions of the directory underlying the file descriptor. If the access mode
47753 is O_SEARCH, the function shall not perform the check.

47754 If *mknodat*( ) is passed the special value AT_FDCWD in the *fd* parameter, the current working
47755 directory shall be used and the behavior shall be identical to a call to *mknod*( ).

47756 **RETURN VALUE**

47757 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
47758 return –1 and set *errno* to indicate the error. If –1 is returned, the new file shall not be created.

47759 **ERRORS**

47760 These functions shall fail if:

47761 [EACCES] A component of the path prefix denies search permission, or write permission
47762 is denied on the parent directory.

47763 [EEXIST] The named file exists.

47764 [EILSEQ] The last pathname component of *path* is not a portable filename, and cannot be
47765 created in the target directory.

47766 [EINVAL] An invalid argument exists.

47767 [EIO] An I/O error occurred while accessing the file system.

47768 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
47769 argument.

47770 [ENAMETOOLONG]
47771 The length of a component of a pathname is longer than {NAME_MAX}.

47772 [ENOENT] A component of the path prefix of *path* does not name an existing file or *path* is
47773 an empty string.

47774 [ENOENT] or [ENOTDIR]
47775 The *path* argument contains at least one non-<slash> character and ends with
47776 one or more trailing <slash> characters. If *path* without the trailing <slash>
47777 characters would name an existing file, an [ENOENT] error shall not occur.

47778 [ENOSPC] The directory that would contain the new file cannot be extended or the file
47779 system is out of file allocation resources.

47780 [ENOTDIR] A component of the path prefix names an existing file that is neither a
47781 directory nor a symbolic link to a directory.

47782 [EPERM] The invoking process does not have appropriate privileges and the file type is
47783 not FIFO-special.

47784 [EROFS] The directory in which the file is to be created is located on a read-only file
47785 system.

47786 The *mknodat*( ) function shall fail if:

47787 [EACCES] The access mode of the open file description associated with *fd* is not
47788 O_SEARCH and the permissions of the directory underlying *fd* do not permit
47789 directory searches.

47790 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
47791 neither AT_FDCWD nor a valid file descriptor open for reading or searching.

47792 [ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated
47793 with a non-directory file.

47794 These functions may fail if:

47795 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
47796 resolution of the *path* argument.

47797 [ENAMETOOLONG]
47798 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
47799 symbolic link produced an intermediate result with a length that exceeds
47800 {PATH_MAX}.

47801 **EXAMPLES**

47802 **Creating a FIFO Special File**

47803 The following example shows how to create a FIFO special file named **/home/cnd/mod_done**,
47804 with read/write permissions for owner, and with read permissions for group and others.

```
47805 #include <sys/types.h>
47806 #include <sys/stat.h>

47807 dev_t dev;
47808 int   status;
47809 ...
47810 status  = mknod("/home/cnd/mod_done", S_IFIFO | S_IWUSR |
47811     S_IRUSR | S_IRGRP | S_IROTH, dev);
```

47812 **APPLICATION USAGE**

47813 The *mkfifo*( ) function is preferred over this function for making FIFO special files.

47814 **RATIONALE**

47815 The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group
47816 ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required
47817 that implementations provide a way to have the group ID be set to the group ID of the
47818 containing directory, but did not prohibit implementations also supporting a way to set the
47819 group ID to the effective group ID of the creating process. Conforming applications should not
47820 assume which group ID will be used. If it matters, an application can use *chown*( ) to set the

group ID after the file is created, or determine under what conditions the implementation will set the desired group ID.

The purpose of the *mknodat*( ) function is to create directories, special files, or regular files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *mknod*( ), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *mknodat*( ) function it can be guaranteed that the newly created directory, special file, or regular file is located relative to the desired directory.

Implementations are encouraged to have *mknod*( ) and *mknodat*( ) report an [EILSEQ] error if the last component of *path* contains any bytes that have the encoded value of a <newline> character.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *creat*( ), *exec*, *fstatat*( ), *mkdir*( ), *mkfifo*( ), *open*( ), *umask*( )

XBD **<fcntl.h>**, **<sys/stat.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

The *mknodat*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

Changes are made to allow a directory to be opened for searching.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0388 [324], XSH/TC1-2008/0389 [461], XSH/TC1-2008/0390 [146,435], XSH/TC1-2008/0391 [278], and XSH/TC1-2008/0392 [278] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0222 [591], XSH/TC2-2008/0223 [817], XSH/TC2-2008/0224 [822], XSH/TC2-2008/0225 [817], and XSH/TC2-2008/0226 [591] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 293 is applied, adding the [EILSEQ] error.

47860 **NAME**

47861 mkostemp — create a unique file

47862 **SYNOPSIS**

47863 CX
```
#include <stdlib.h>

int mkostemp(char *template, int flag);
```
47864

47865 **DESCRIPTION**

47866 Refer to *mkdtemp( )*.

47867 **NAME**

47868 mkstemp — create a unique file

47869 **SYNOPSIS**

47870 CX
```
#include <stdlib.h>

int mkstemp(char *template);
```
47871

47872 **DESCRIPTION**

47873 Refer to *mkdtemp( )*.

NAME

mktime — convert broken-down time into time since the Epoch

SYNOPSIS

```
#include <time.h>

time_t mktime(struct tm *timeptr);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *mktime*( ) function shall convert the broken-down time, expressed as local time, in some members of the structure pointed to by *timeptr*, into a time since the Epoch value with the same encoding as that of the values returned by *time*( ). The *mktime*( ) function shall make use of only the *tm_year*, *tm_mon*, *tm_mday*, *tm_hour*, *tm_min*, *tm_sec*, and *tm_isdst* members of the structure pointed to by *timeptr*; the values of these members shall not be restricted to the ranges described in **<time.h>**.

CX Local timezone information shall be set as though *mktime*( ) called *tzset*( ).

The *mktime*( ) function shall calculate the time in seconds since the Epoch to be returned as if by manipulating the members of the **tm** structure according to the following steps.

1. The *tm_sec* member may, but should not, be brought into the range 0 to 60, inclusive. For each 60 seconds added to or subtracted from *tm_sec*, a decrement or increment, respectively, of 1 minute shall be saved for later application.
2. The *tm_min* member shall be brought into the range 0 to 59, inclusive, and any saved decrement or increment of minutes shall then be applied, repeating the range adjustment afterwards if necessary. For each 60 minutes added to or subtracted from *tm_min*, a decrement or increment, respectively, of 1 hour shall be saved for later application.
3. The *tm_hour* member shall be brought into the range 0 to 23, inclusive, and any saved decrement or increment of hours shall then be applied, repeating the range adjustment afterwards if necessary. For each 24 hours added to or subtracted from *tm_hour*, a decrement or increment, respectively, of 1 day shall be saved for later application.
4. The *tm_mon* member shall be brought into the range 0 to 11, inclusive. For each 12 months added to or subtracted from *tm_mon*, a decrement or increment, respectively, of 1 year shall be saved for later use.
5. The *tm_mday* member shall be brought into the range 1 to 31, inclusive, and any saved decrement or increment of days shall then be applied, repeating the range adjustment afterwards if necessary. Adjustments downwards shall be applied by subtracting the number of days (according to the Gregorian calendar) in month *tm_mon*+1 of the year obtained by adding/subtracting any saved increment/decrement of years to the value *tm_year*+1900, and then incrementing *tm_mon* by 1, repeated as necessary. Adjustments upwards shall be applied by adding the number of days in the month before month *tm_mon*+1 of the year obtained by adding/subtracting any saved increment/decrement of years to the value *tm_year*+1900, and then decrementing *tm_mon* by 1, repeated as necessary. During these adjustments, the *tm_mon* value shall be kept within the range 0 to 11, inclusive, by applying step 4 as necessary.
6. If the *tm_mday* member is greater than the number of days in month *tm_mon*+1 of the year obtained by adding/subtracting any saved increment/decrement of years to the value *tm_year*+1900, that number of days shall be subtracted from *tm_mday*, and *tm_mon* shall

be incremented by 1. If this results in *tm_mon* having the value 12, step 4 shall be applied.

7. The number of seconds since the Epoch in Coordinated Universal Time shall be calculated from the range-corrected values of the relevant **tm** structure members (or the original value where a member was not range corrected) as specified in the expression given in the definition of seconds since the Epoch (see XBD Section 4.19, on page 107), where the names other than *tm_year* and *tm_yday* in the structure and in the expression correspond, the *tm_year* value used in the expression is the *tm_year* in the structure plus/minus any saved increment/decrement of years, and the *tm_yday* value used in the expression is the day of the year from 0 to 365 inclusive, calculated from the *tm_mon* and *tm_mday* members of the **tm** structure, for that year.
8. The time since the Epoch shall be corrected for the offset of the local timezone's standard time from Coordinated Universal Time.
9. The time since the Epoch shall be further corrected (if applicable—see below) for Daylight Saving Time.

If the timezone is one that includes Daylight Saving Time (DST) adjustments, the value of *tm_isdst* in the **tm** structure controls whether or not *mktime*( ) adjusts the calculated seconds since the Epoch value by the DST offset (after it has made the timezone adjustment), as follows:

- If *tm_isdst* is zero, *mktime*( ) shall not further adjust the seconds since the Epoch by the DST offset.
- If *tm_isdst* is positive, *mktime*( ) shall further adjust the seconds since the Epoch by the DST offset.
- If *tm_isdst* is negative, *mktime*( ) shall attempt to determine whether DST is in effect for the specified time; if it determines that DST is in effect it shall produce the same result as an equivalent call with a positive *tm_isdst* value, otherwise it shall produce the same result as an equivalent call with a *tm_isdst* value of zero. If the broken-down time specifies a time that is either skipped over or repeated when a transition to or from DST occurs, it is unspecified whether *mktime*( ) produces the same result as an equivalent call with a positive *tm_isdst* value or as an equivalent call with a *tm_isdst* value of zero.

If the *TZ* environment variable specifies a geographical timezone for which the implementation's timezone database includes historical or future changes to the offset from Coordinated Universal Time of the timezone's standard time, and the broken-down time corresponds to a time that was (or will be) skipped over or repeated due to the occurrence of such a change, *mktime*( ) shall calculate the time since the Epoch value using either the offset in effect before the change or the offset in effect after the change.

Upon successful completion, the members of the structure shall be set to the values that would be returned by a call to *localtime*( ) with the calculated time since the Epoch as its argument.

RETURN VALUE

The *mktime*( ) function shall return the calculated time since the Epoch encoded as a value of type **time_t**. If the time since the Epoch cannot be represented as a **time_t** or the value to be returned in the *tm_year* member of the structure pointed to by *timeptr* cannot be represented as an **int**, the function shall return the value (**time_t**)–1 and set *errno* to [EOVERFLOW], and shall not change the value of the *tm_wday* component of the structure.

Since (**time_t**)–1 is a valid return value for a successful call to *mktime*( ), an application wishing to check for error situations should set *tm_wday* to a value less than 0 or greater than 6 before calling *mktime*( ). On return, if *tm_wday* has not changed an error has occurred.

ERRORS

The *mktime*( ) function shall fail if:

CX [EOVERFLOW] The result cannot be represented.

EXAMPLES

What day of the week is July 4, 2001?

```
#include <stdio.h>
#include <time.h>

struct tm time_str;

char daybuf[20];

int main(void)
{
    time_str.tm_year = 2001 - 1900;
    time_str.tm_mon = 7 - 1;
    time_str.tm_mday = 4;
    time_str.tm_hour = 0;
    time_str.tm_min = 0;
    time_str.tm_sec = 1;
    time_str.tm_isdst = -1;
    time_str.tm_wday = -1;
    if (mktime(&time_str) == (time_t)-1 && time_str.tm_wday == -1)
        (void)puts("-unknown-");
    else {
        (void)strftime(daybuf, sizeof(daybuf), "%A", &time_str);
        (void)puts(daybuf);
    }
    return 0;
}
```

APPLICATION USAGE

When using *mktime*( ) to add or subtract a fixed time period (one that always corresponds to a fixed number of seconds) to or from a broken-down time in the local timezone, reliable results for arbitrary *TZ* can only be assured by using *mktime*( ) to convert the original broken-down time to a time since the Epoch, adding or subtracting the desired number of seconds to that value, and then calling *localtime*( ) with the result. The alternative of adjusting the broken-down time before calling *mktime*( ) may produce unexpected results if the original and updated times are on different sides of a geographical timezone change. On implementations that follow the recommendation of not range-correcting *tm_sec* (see step 1 in the DESCRIPTION), reliable results can also be assured by adding or subtracting the desired number of seconds to *tm_sec* (and not modifying any other members of the **tm** structure). In applications needing to be portable to non-POSIX systems where the **time_t** encoding is not a count of seconds, it is recommended that conditional compilation is used such that the adjustment is performed on the *mktime*( ) return value when possible, and otherwise on the *tm_sec* member. For timezones that are known not to have geographical timezone changes, such as `TZ=UTC0`, adjustments using just *mktime*( ) do not have this problem.

The way the *mktime*( ) function interprets out-of-range **tm** structure fields might not produce the expected result when multiple adjustments are made at the same time. For example, if an application tries to go back one day first and then one year by calling *localtime*( ), decrementing *tm_mday* and *tm_year*, and then calling *mktime*( ) this would not produce the expected result if it was called on 2021–03–01 because *mktime*( ) would see the supplied year as 2020 (a leap year)

and correct Mar 0 to Feb 29, whereas the intended result was Feb 28. Such issues can be avoided by doing multiple adjustments one at a time when the order in which they are done matters.

Examples of how *mktime*( ) handles some adjustments are:

- If given Feb 29 in a non-leap year it treats that as the day after Feb 28 and gives back Mar 1.
- If given Feb 0 it treats that as the day before Feb 1 and gives back Jan 31.
- If given 21:65 it treats that as 6 minutes after 21:59 and gives back 22:05.
- If given *tm_isdst*=0 for a time when DST is in effect, it gives back a positive *tm_isdst* and alters the other fields appropriately.
- If there is a DST transition where 02:00 standard time becomes 03:00 DST and *mktime*( ) is given 02:30 (with negative *tm_isdst*), it treats that as either 30 minutes after 02:00 standard time or 30 minutes before 03:00 DST and gives back a zero or positive *tm_isdst*, respectively, with the *tm_hour* field altered appropriately.
- If a geographical timezone changes its UTC offset such that ``old 00:00'' becomes ``new 00:30'' and *mktime*( ) is given 00:20, it treats that as either 20 minutes after ``old 00:00'' or 10 minutes before ``new 00:30'', and gives back appropriately altered **struct tm** fields.

If an application wants to check whether a given broken-down time is one that is skipped over, it can do so by seeing whether the *tm_mday*, *tm_hour*, and *tm_min* values it gets back from *mktime*( ) are the same ones it fed in. Just checking *tm_hour* and *tm_min* might appear at first sight to suffice, but *tm_mday* could also change—without *tm_hour* and *tm_min* changing—if, for example, *TZ* is set to `"ABC12XYZ-12"` (which might be used in a torture test) or if a geographical timezone changes the offset from Coordinated Universal Time of its standard time by 24 hours.

RATIONALE

In order to allow applications to distinguish between a successful return of (**time_t**)-1 and an [EOVERFLOW] error, *mktime*( ) is required not to change *tm_wday* on error. This mechanism is used rather than the convention used for other functions whereby the application sets *errno* to zero before the call and the call does not change *errno* on error because the ISO C standard does not require *mktime*( ) to set *errno* on error. The next revision of the ISO C standard is expected to require that *mktime*( ) does not change *tm_wday* when returning (**time_t**)-1 to indicate an error, and that this return convention is used both for the case where the value to be returned by the function cannot be represented as a **time_t** and the case where the value to be returned in the *tm_year* member of the **tm** structure cannot be represented as an **int**.

The DESCRIPTION section says that *mktime*( ) converts the specified broken-down time into **a** time since the Epoch value. The use of the indefinite article here is necessary because, when *tm_isdst* is negative and the timezone has Daylight Saving Time transitions, there is not a one-to-one correspondence between broken-down times and time since the Epoch values.

The description of how the value of *tm_isdst* affects the behavior of *mktime*( ) is shaded CX because the requirements in the ISO C standard are unclear. The next revision of the ISO C standard is expected to state the requirements using wording equivalent to the wording in this standard.

Implementations are encouraged not to range-correct *tm_sec* (see step 1 in the DESCRIPTION) in order for the results of making an adjustment to *tm_sec* always to be equivalent to making the same adjustment to the value returned by *mktime*( ), even when the original and updated times are on different sides of a geographical timezone change. This provides a way for applications to do reliable fixed-period adjustment using only *mktime*( ), as described in APPLICATION USAGE.

The described method for range-correcting the **tm** structure members uses separate variables to hold adjustment values to be applied later to other members, or (for the year adjustment) used in later calculations, because this is one way of avoiding intermediate member values that are not representable as an **int**. Implementations may use other methods; all that is required is that *tm_year* is the only member for which an [EOVERFLOW] error can occur.

The described method for range-correcting *tm_mday* would, if implemented that way, be highly inefficient for very large values. The efficiency can be improved by observing that any period of 400 years always has the same number of days, so the month-by-month correction method need only be applied for a maximum of 4800 months.

FUTURE DIRECTIONS

A future version of this standard may require that *mktime*( ) does not perform the optional range correction of the *tm_sec* member of the **tm** structure described at step 1 in the DESCRIPTION.

A future version of this standard is expected to add a *timegm*( ) function that is similar to *mktime*( ), except that the **tm** structure pointed to by *timeptr* contains a broken-down time in Coordinated Universal Time (rather than the local timezone), where references to *localtime*( ) are replaced by references to *gmtime*( ), and where there are no timezone offset or Daylight Saving Time adjustments. A combination of *gmtime*( ) and *timegm*( ) will be the expected way to perform arithmetic upon a **time_t** value and remain compatible with the ISO C standard (where the internal structure of a **time_t** is not specified), since attempting such manipulations using *localtime*( ) and *mktime*( ) can lead to unexpected results.

SEE ALSO

asctime( ), *clock*( ), *ctime*( ), *difftime*( ), *futimens*( ), *gmtime*( ), *localtime*( ), *strftime*( ), *strptime*( ), *time*( ), *tzset*( )

XBD Section 4.19 (on page 107), **<time.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ANSI C standard.

Issue 6

Extensions beyond the ISO C standard are marked.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/58 is applied, updating the RETURN VALUE and ERRORS sections to add the optional [EOVERFLOW] error as a CX extension.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/59 is applied, adding the *tzset*( ) function to the SEE ALSO section.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0393 [104] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0228 [724] is applied.

Issue 8

Austin Group Defect 1253 is applied, changing ``Daylight Savings'' to ``Daylight Saving''.

Austin Group Defect 1613 is applied, changing the way the **tm** structure members used by *mktime*( ) are specified and clarifying that a successful call sets the members to the same values that would be returned by *localtime*( ).

Austin Group Defect 1614 is applied, clarifying how *tm_isdst* is handled and the conditions under which (**time_t**)–1 is returned.

Austin Group Defect 1627 is applied, clarifying how *mktime*( ) calculates the time in seconds

48102 since the Epoch from the members of the **tm** structure.

NAME

mlock, munlock — lock or unlock a range of process address space (**REALTIME**)

SYNOPSIS

MLR

```
#include <sys/mman.h>

int mlock(const void *addr, size_t len);
int munlock(const void *addr, size_t len);
```

DESCRIPTION

The *mlock*( ) function shall cause those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes to be memory-resident until unlocked or until the process exits or *exec*s another process image. The implementation may require that *addr* be a multiple of {PAGESIZE}.

The *munlock*( ) function shall unlock those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes, regardless of how many times *mlock*( ) has been called by the process for any of the pages in the specified range. The implementation may require that *addr* be a multiple of {PAGESIZE}.

If any of the pages in the range specified to a call to *munlock*( ) are also mapped into the address spaces of other processes, any locks established on those pages by another process are unaffected by the call of this process to *munlock*( ). If any of the pages in the range specified by a call to *munlock*( ) are also mapped into other portions of the address space of the calling process outside the range specified, any locks established on those pages via the other mappings are also unaffected by this call.

Upon successful return from *mlock*( ), pages in the specified range shall be locked and memory-resident. Upon successful return from *munlock*( ), pages in the specified range shall be unlocked with respect to the address space of the process. Memory residency of unlocked pages is unspecified.

Appropriate privileges are required to lock process memory with *mlock*( ).

RETURN VALUE

Upon successful completion, the *mlock*( ) and *munlock*( ) functions shall return a value of zero. Otherwise, no change is made to any locks in the address space of the process, and the function shall return a value of –1 and set *errno* to indicate the error.

ERRORS

The *mlock*( ) and *munlock*( ) functions shall fail if:

[ENOMEM] Some or all of the address range specified by the *addr* and *len* arguments does not correspond to valid mapped pages in the address space of the process.

The *mlock*( ) function shall fail if:

[EAGAIN] Some or all of the memory identified by the operation could not be locked when the call was made.

The *mlock*( ) and *munlock*( ) functions may fail if:

[EINVAL] The *addr* argument is not a multiple of {PAGESIZE}.

The *mlock*( ) function may fail if:

[ENOMEM] Locking the pages mapped by the specified range would exceed an implementation-defined limit on the amount of memory that the process may lock.

48146 [EPERM] The calling process does not have appropriate privileges to perform the
48147 requested operation.

48148 **EXAMPLES**

48149 None.

48150 **APPLICATION USAGE**

48151 None.

48152 **RATIONALE**

48153 None.

48154 **FUTURE DIRECTIONS**

48155 None.

48156 **SEE ALSO**

48157 *exec*, *exit*( ), *fork*( ), *mlockall*( ), *munmap*( )

48158 XBD **<sys/mman.h>**

48159 **CHANGE HISTORY**

48160 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

48161 **Issue 6**

48162 The *mlock*( ) and *munlock*( ) functions are marked as part of the Range Memory Locking option.

48163 The [ENOSYS] error condition has been removed as stubs need not be provided if an
48164 implementation does not support the Range Memory Locking option.

NAME

mlockall, munlockall — lock/unlock the address space of a process (**REALTIME**)

SYNOPSIS

ML
```
#include <sys/mman.h>

int mlockall(int flags);
int munlockall(void);
```

DESCRIPTION

The *mlockall*( ) function shall cause all of the pages mapped by the address space of a process to be memory-resident until unlocked or until the process exits or *exec*s another process image. The *flags* argument determines whether the pages to be locked are those currently mapped by the address space of the process, those that are mapped in the future, or both. The *flags* argument is constructed from the bitwise-inclusive OR of one or more of the following symbolic constants, defined in **<sys/mman.h>**:

MCL_CURRENT Lock all of the pages currently mapped into the address space of the process.

MCL_FUTURE Lock all of the pages that become mapped into the address space of the process in the future, when those mappings are established.

If MCL_FUTURE is specified, and the automatic locking of future mappings eventually causes the amount of locked memory to exceed the amount of available physical memory or any other implementation-defined limit, the behavior is implementation-defined. The manner in which the implementation informs the application of these situations is also implementation-defined.

The *munlockall*( ) function shall unlock all currently mapped pages of the address space of the process. Any pages that become mapped into the address space of the process after a call to *munlockall*( ) shall not be locked, unless there is an intervening call to *mlockall*( ) specifying MCL_FUTURE or a subsequent call to *mlockall*( ) specifying MCL_CURRENT. If pages mapped into the address space of the process are also mapped into the address spaces of other processes and are locked by those processes, the locks established by the other processes shall be unaffected by a call by this process to *munlockall*( ).

Upon successful return from the *mlockall*( ) function that specifies MCL_CURRENT, all currently mapped pages of the address space of the process shall be memory-resident and locked. Upon return from the *munlockall*( ) function, all currently mapped pages of the address space of the process shall be unlocked with respect to the address space of the process. The memory residency of unlocked pages is unspecified.

Appropriate privileges are required to lock process memory with *mlockall*( ).

RETURN VALUE

Upon successful completion, the *mlockall*( ) function shall return a value of zero. Otherwise, no additional memory shall be locked, and the function shall return a value of −1 and set *errno* to indicate the error. The effect of failure of *mlockall*( ) on previously existing locks in the address space is unspecified.

If it is supported by the implementation, the *munlockall*( ) function shall always return a value of zero. Otherwise, the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *mlockall*( ) function shall fail if:

[EAGAIN] Some or all of the memory identified by the operation could not be locked when the call was made.

48209 [EINVAL] The *flags* argument is zero, or includes unimplemented flags.

48210 The *mlockall*( ) function may fail if:

48211 [ENOMEM] Locking all of the pages currently mapped into the address space of the
48212 process would exceed an implementation-defined limit on the amount of
48213 memory that the process may lock.

48214 [EPERM] The calling process does not have appropriate privileges to perform the
48215 requested operation.

48216 **EXAMPLES**

48217 None.

48218 **APPLICATION USAGE**

48219 None.

48220 **RATIONALE**

48221 None.

48222 **FUTURE DIRECTIONS**

48223 None.

48224 **SEE ALSO**

48225 *exec*, *exit*( ), *fork*( ), *mlock*( ), *munmap*( )

48226 XBD **<sys/mman.h>**

48227 **CHANGE HISTORY**

48228 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

48229 **Issue 6**

48230 The *mlockall*( ) and *munlockall*( ) functions are marked as part of the Process Memory Locking
48231 option.

48232 The [ENOSYS] error condition has been removed as stubs need not be provided if an
48233 implementation does not support the Process Memory Locking option.

48234 **NAME**

48235 mmap — map pages of memory

48236 **SYNOPSIS**

48237
```
#include <sys/mman.h>
```

48238
```
void *mmap(void *addr, size_t len, int prot, int flags,
```
48239
```
    int fildes, off_t off);
```

48240 **DESCRIPTION**

48241 The *mmap*( ) function shall establish a mapping between an address space of a process and a
48242 memory object.

48243 The *mmap*( ) function shall be supported for the following memory objects:

48244 • Regular files

48245 • Anonymous memory objects

48246 SHM • Shared memory objects

48247 TYM • Typed memory objects

48248 Support for any other type of file is unspecified.

48249 The format of the call is as follows:

48250
```
pa=mmap(addr, len, prot, flags, fildes, off);
```

48251 The *mmap*( ) function shall establish a mapping between the address space of the process at an
48252 address *pa* for *len* bytes to the memory object represented by the file descriptor *fildes* at offset *off*
48253 for *len* bytes, or to an anonymous memory object of *len* bytes. The value of *pa* is an
48254 implementation-defined function of the parameter *addr* and the values of *flags*, further described
48255 below. A successful *mmap*( ) call shall return *pa* as its result. The address range starting at *pa* and
48256 continuing for *len* bytes shall be legitimate for the possible (not necessarily current) address
48257 space of the process. The range of bytes starting at *off* and continuing for *len* bytes shall be
48258 legitimate for the possible (not necessarily current) offsets in the memory object represented by
48259 *fildes*.

48260 TYM If *fildes* represents a typed memory object opened with either the
48261 POSIX_TYPED_MEM_ALLOCATE flag or the POSIX_TYPED_MEM_ALLOCATE_CONTIG
48262 flag, the memory object to be mapped shall be that portion of the typed memory object allocated
48263 by the implementation as specified below. In this case, if *off* is non-zero, the behavior of *mmap*( )
48264 is undefined. If *fildes* refers to a valid typed memory object that is not accessible from the calling
48265 process, *mmap*( ) shall fail.

48266 The mapping established by *mmap*( ) shall replace any previous mappings for those whole pages
48267 containing any part of the address space of the process starting at *pa* and continuing for *len*
48268 bytes.

48269 If the size of the mapped file changes after the call to *mmap*( ) as a result of some other operation
48270 on the mapped file, the effect of references to portions of the mapped region that correspond to
48271 added or removed portions of the file is unspecified.

48272 If *len* is zero, *mmap*( ) shall fail and no mapping shall be established.

48273 The parameter *prot* determines whether read, write, execute, or some combination of accesses
48274 are permitted to the data being mapped. The *prot* shall be either PROT_NONE or the bitwise-
48275 inclusive OR of one or more of the other flags in the following table, defined in the
48276 **<sys/mman.h>** header.

Symbolic Constant	Description
PROT_READ	Data can be read.
PROT_WRITE	Data can be written.
PROT_EXEC	Data can be executed.
PROT_NONE	Data cannot be accessed.

If an implementation cannot support the combination of access types specified by *prot*, the call to *mmap*( ) shall fail.

An implementation may permit accesses other than those specified by *prot*; however, the implementation shall not permit a write to succeed where PROT_WRITE has not been set and shall not permit any access where PROT_NONE alone has been set. The implementation shall support at least the following values of *prot*: PROT_NONE, PROT_READ, PROT_WRITE, and the bitwise-inclusive OR of PROT_READ and PROT_WRITE. The file descriptor *fildes* shall have been opened with read permission, regardless of the protection options specified. If PROT_WRITE is specified, the application shall ensure that it has opened the file descriptor *fildes* with write permission unless MAP_PRIVATE is specified in the *flags* parameter as described below.

The parameter *flags* provides other information about the handling of the mapped data. The value of *flags* is the bitwise-inclusive OR of these options, defined in **<sys/mman.h>**:

Symbolic Constant	Description
MAP_ANON	Synonym for MAP_ANONYMOUS.
MAP_ANONYMOUS	Map anonymous memory.
MAP_SHARED	Changes are shared.
MAP_PRIVATE	Changes are private.
MAP_FIXED	Interpret *addr* exactly.

XSI It is implementation-defined whether MAP_FIXED shall be supported. MAP_FIXED shall be supported on XSI-conformant systems.

MAP_SHARED and MAP_PRIVATE describe the disposition of write references to the memory object. If MAP_SHARED is specified, write references shall change the underlying object. If MAP_PRIVATE is specified, modifications to the mapped data by the calling process shall be visible only to the calling process and shall not change the underlying object. It is unspecified whether modifications to the underlying object done after the MAP_PRIVATE mapping is established are visible through the MAP_PRIVATE mapping. Either MAP_SHARED or MAP_PRIVATE can be specified, but not both. The mapping type is retained across *fork*( ).

The state of synchronization objects such as mutexes, semaphores, barriers, and conditional variables placed in shared memory mapped with MAP_SHARED becomes undefined when the last region in any process containing the synchronization object is unmapped.

TYM When *fildes* represents a typed memory object opened with either the POSIX_TYPED_MEM_ALLOCATE flag or the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, *mmap*( ) shall, if there are enough resources available, map *len* bytes allocated from the corresponding typed memory object which were not previously allocated to any process in any processor that may access that typed memory object. If there are not enough resources available, the function shall fail. If *fildes* represents a typed memory object opened with the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, these allocated bytes shall be contiguous within the typed memory object. If *fildes* represents a typed memory object opened with the POSIX_TYPED_MEM_ALLOCATE flag, these allocated bytes may be composed of non-contiguous fragments within the typed memory object. If *fildes* represents a typed memory object opened with neither the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag nor the

POSIX_TYPED_MEM_ALLOCATE flag, *len* bytes starting at offset *off* within the typed memory object are mapped, exactly as when mapping a file or shared memory object. In this case, if two processes map an area of typed memory using the same *off* and *len* values and using file descriptors that refer to the same memory pool (either from the same port or from a different port), both processes shall map the same region of storage.

When MAP_FIXED is set in the *flags* argument, the implementation is informed that the value of *pa* shall be *addr*, exactly. If MAP_FIXED is set, *mmap*( ) may return MAP_FAILED and set *errno* to ML|MLR [EINVAL]. If a MAP_FIXED request is successful, then any previous mappings or memory locks for those whole pages containing any part of the address range [*pa*,*pa*+*len*) shall be removed, as if by an appropriate call to *munmap*( ), before the new mapping is established.

When MAP_FIXED is not set, the implementation uses *addr* in an implementation-defined manner to arrive at *pa*. The *pa* so chosen shall be an area of the address space that the implementation deems suitable for a mapping of *len* bytes to the file. All implementations interpret an *addr* value of 0 as granting the implementation complete freedom in selecting *pa*, subject to constraints described below. A non-zero value of *addr* is taken to be a suggestion of a process address near which the mapping should be placed. When the implementation selects a value for *pa*, it never places a mapping at address 0, nor does it replace any extant mapping.

If MAP_FIXED is specified and *addr* is non-zero, it shall have the same remainder as the *off* parameter, modulo the page size as returned by *sysconf*( ) when passed _SC_PAGESIZE or _SC_PAGE_SIZE. The implementation may require that off is a multiple of the page size. If MAP_FIXED is specified, the implementation may require that *addr* is a multiple of the page size. The system performs mapping operations over whole pages. Thus, while the parameter *len* need not meet a size or alignment constraint, the system shall include, in any mapping operation, any partial page specified by the address range starting at *pa* and continuing for *len* bytes.

If MAP_ANONYMOUS (or its synonym MAP_ANON) is specified, *fildes* is –1, and *off* is 0, then *mmap*( ) shall ignore *fildes* and instead establish a mapping to a new anonymous memory object of size *len*. The effect of specifying MAP_ANONYMOUS (or MAP_ANON) with other values of *fildes* or *off* is unspecified. Anonymous memory objects shall be initialized to all bits zero.

The system shall always zero-fill any partial page at the end of an object. Further, the system shall never write out any modified portions of the last page of an object which are beyond its end. References within the address range starting at *pa* and continuing for *len* bytes to whole pages following the end of an object shall result in delivery of a SIGBUS signal.

An implementation may generate SIGBUS signals when a reference would cause an error in the mapped object, such as out-of-space condition.

The *mmap*( ) function shall add an extra reference to the file associated with the file descriptor *fildes* which is not removed by a subsequent *close*( ) on that file descriptor. This reference shall be removed when there are no more mappings to the file.

The last data access timestamp of the mapped file may be marked for update at any time between the *mmap*( ) call and the corresponding *munmap*( ) call. The initial read or write reference to a mapped region shall cause the file's last data access timestamp to be marked for update if it has not already been marked for update.

The last data modification and last file status change timestamps of a file that is mapped with MAP_SHARED and PROT_WRITE shall be marked for update at some point in the interval between a write reference to the mapped region and the next call to *msync*( ) with MS_ASYNC or MS_SYNC for that portion of the file by any process. If there is no such call and if the underlying file is modified as a result of a write reference, then these timestamps shall be

marked for update at some time after the write reference.

There may be implementation-defined limits on the number of memory regions that can be mapped (per process or per system).

XSI If such a limit is imposed, whether the number of memory regions that can be mapped by a process is decreased by the use of *shmat*( ) is implementation-defined.

If *mmap*( ) fails for reasons other than [EBADF], [EINVAL], or [ENOTSUP], some of the mappings in the address range starting at *addr* and continuing for *len* bytes may have been unmapped.

RETURN VALUE

Upon successful completion, the *mmap*( ) function shall return the address at which the mapping was placed (*pa*); otherwise, it shall return a value of MAP_FAILED and set *errno* to indicate the error. The symbol MAP_FAILED is defined in the **<sys/mman.h>** header. No successful return from *mmap*( ) shall return the value MAP_FAILED.

ERRORS

The *mmap*( ) function shall fail if:

ML [EAGAIN] The mapping could not be locked in memory, if required by *mlockall*( ), due to a lack of resources.

[EINVAL] The value of *len* is zero.

[EINVAL] The value of *flags* is invalid (neither MAP_PRIVATE nor MAP_SHARED is set).

[EMFILE] The number of mapped regions would exceed an implementation-defined limit (per process or per system).

[ENOMEM] MAP_FIXED was specified, and the range [*addr*,*addr*+*len*) exceeds that allowed for the address space of a process; or, if MAP_FIXED was not specified and there is insufficient room in the address space to effect the mapping.

ML [ENOMEM] The mapping could not be locked in memory, if required by *mlockall*( ), because it would require more space than the system is able to supply.

TYM [ENOMEM] Not enough unallocated memory resources remain in the typed memory object designated by *fildes* to allocate *len* bytes.

[ENOTSUP] MAP_FIXED or MAP_PRIVATE was specified in the *flags* argument and the implementation does not support this functionality.

The implementation does not support the combination of accesses requested in the *prot* argument.

[ENXIO] MAP_FIXED was specified in *flags* and the combination of *addr*, *len*, and *off* is invalid for the specified object.

TYM [ENXIO] The *fildes* argument refers to a typed memory object that is not accessible from the calling process.

In addition, if MAP_ANONYMOUS (or MAP_ANON) is not set in flags, the *mmap*( ) function shall fail if:

[EACCES] The *fildes* argument is not open for read, regardless of the protection specified, or *fildes* is not open for write and PROT_WRITE was specified for a MAP_SHARED type mapping.

[EBADF] The *fildes* argument is not a valid open file descriptor.

[ENODEV] The *fildes* argument refers to a file whose type is not supported by *mmap*( ).

[EOVERFLOW] The file is a regular file and the value of *off* plus *len* exceeds the offset maximum established in the open file description associated with *fildes*.

[ENXIO] Addresses in the range [*off*,*off*+*len*) are invalid for the object specified by *fildes*.

The *mmap*( ) function may fail if:

[EINVAL] The *addr* argument (if MAP_FIXED was specified) or *off* is not a multiple of the page size as returned by *sysconf*( ), or is considered invalid by the implementation.

EXAMPLES

None.

APPLICATION USAGE

Use of *mmap*( ) may reduce the amount of memory available to other memory allocation functions.

Use of MAP_FIXED may result in unspecified behavior in further use of *malloc*( ) and *shmat*( ). The use of MAP_FIXED is discouraged, as it may prevent an implementation from making the most effective use of resources. Most implementations require that *off* and *addr* are multiples of the page size as returned by *sysconf*( ).

The application must ensure correct synchronization when using *mmap*( ) in conjunction with any other file access method, such as *read*( ) and *write*( ), standard input/output, and *shmat*( ).

The *mmap*( ) function allows access to resources via address space manipulations, instead of *read*( )/*write*( ). Once a file is mapped, all a process has to do to access it is use the data at the address to which the file was mapped. So, using pseudo-code to illustrate the way in which an existing program might be changed to use *mmap*( ), the following:

```
fildes = open(...)
lseek(fildes, some_offset)
read(fildes, buf, len)
/* Use data in buf. */
```

becomes:

```
fildes = open(...)
address = mmap(0, len, PROT_READ, MAP_PRIVATE, fildes, some_offset)
/* Use data at address. */
```

RATIONALE

After considering several other alternatives, it was decided to adopt the *mmap*( ) definition found in SVR4 for mapping memory objects into process address spaces. The SVR4 definition is minimal, in that it describes only what has been built, and what appears to be necessary for a general and portable mapping facility.

Note that while *mmap*( ) was first designed for mapping files, it is actually a general-purpose mapping facility. It can be used to map any appropriate object, such as memory, files, devices, and so on, into the address space of a process.

When a mapping is established, it is possible that the implementation may need to map more than is requested into the address space of the process because of hardware requirements. An application, however, cannot count on this behavior. Implementations that do not use a paged architecture may simply allocate a common memory region and return the address of it; such

implementations probably do not allocate any more than is necessary. References past the end of the requested area are unspecified.

If an application requests a mapping that overlaps existing mappings in the process, it might be desirable that an implementation detect this and inform the application. However, if the program specifies a fixed address mapping (which requires some implementation knowledge to determine a suitable address, if the function is supported at all), then the program is presumed to be successfully managing its own address space and should be trusted when it asks to map over existing data structures. Furthermore, it is also desirable to make as few system calls as possible, and it might be considered onerous to require an *munmap*( ) before an *mmap*( ) to the same address range. This volume of POSIX.1-2024 specifies that the new mapping replaces any existing mappings (implying an automatic *munmap*( ) on the address range), following existing practice in this regard. The standard developers also considered whether there should be a way for new mappings to overlay existing mappings, but found no existing practice for this.

It is not expected that all hardware implementations are able to support all combinations of permissions at all addresses. Implementations are required to disallow write access to mappings without write permission and to disallow access to mappings without any access permission. Other than these restrictions, implementations may allow access types other than those requested by the application. For example, if the application requests only PROT_WRITE, the implementation may also allow read access. A call to *mmap*( ) fails if the implementation cannot support allowing all the access requested by the application. For example, some implementations cannot support a request for both write access and execute access simultaneously. All implementations must support requests for no access, read access, write access, and both read and write access. Strictly conforming code must only rely on the required checks. These restrictions allow for portability across a wide range of hardware.

The MAP_FIXED address treatment is likely to fail for non-page-aligned values and for certain architecture-dependent address ranges. Conforming implementations cannot count on being able to choose address values for MAP_FIXED without utilizing non-portable, implementation-defined knowledge. Nonetheless, MAP_FIXED is provided as a standard interface conforming to existing practice for utilizing such knowledge when it is available.

Similarly, in order to allow implementations that do not support virtual addresses, support for directly specifying any mapping addresses via MAP_FIXED is not required and thus a conforming application may not count on it.

The MAP_PRIVATE function can be implemented efficiently when memory protection hardware is available. When such hardware is not available, implementations can implement such ``mappings'' by simply making a real copy of the relevant data into process private memory, though this tends to behave similarly to *read*( ).

The function has been defined to allow for many different models of using shared memory. However, all uses are not equally portable across all machine architectures. In particular, the *mmap*( ) function allows the system as well as the application to specify the address at which to map a specific region of a memory object. The most portable way to use the function is always to let the system choose the address, specifying NULL as the value for the argument *addr* and not to specify MAP_FIXED.

If it is intended that a particular region of a memory object be mapped at the same address in a group of processes (on machines where this is even possible), then MAP_FIXED can be used to pass in the desired mapping address. The system can still be used to choose the desired address if the first such mapping is made without specifying MAP_FIXED, and then the resulting mapping address can be passed to subsequent processes for them to pass in via MAP_FIXED. The availability of a specific address range cannot be guaranteed, in general.

The *mmap*( ) function can be used to map a region of memory that is larger than the current size of the object. Memory access within the mapping but beyond the current end of the underlying objects may result in SIGBUS signals being sent to the process. The reason for this is that the size of the object can be manipulated by other processes and can change at any moment. The implementation should tell the application that a memory reference is outside the object where this can be detected; otherwise, written data may be lost and read data may not reflect actual data in the object.

Note that references beyond the end of the object do not extend the object as the new end cannot be determined precisely by most virtual memory hardware. Instead, the size can be directly manipulated by *ftruncate*( ).

Process memory locking does apply to shared memory regions, and the MCL_FUTURE argument to *mlockall*( ) can be relied upon to cause new shared memory regions to be automatically locked.

Existing implementations of *mmap*( ) return the value –1 when unsuccessful. Since the casting of this value to type **void** * cannot be guaranteed by the ISO C standard to be distinct from a successful value, this volume of POSIX.1-2024 defines the symbol MAP_FAILED, which a conforming implementation does not return as the result of a successful call.

Some historical implementations only supported MAP_ANON, some only supported MAP_ANONYMOUS, and some supported both spellings. This standard includes both spellings partly for application compatibility and partly because neither spelling was clearly more popular than the other at the time this feature was considered for standardization.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fcntl*( ), *fork*( ), *lockf*( ), *msync*( ), *munmap*( ), *mprotect*( ), *posix_typed_mem_open*( ), *shmat*( ), *sysconf*( )

XBD **<sys/mman.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Aligned with *mmap*( ) in the POSIX Realtime Extension as follows:

- The DESCRIPTION is extensively reworded.
- The [EAGAIN] and [ENOTSUP] mandatory error conditions are added.
- New cases of [ENOMEM] and [ENXIO] are added as mandatory error conditions.
- The value returned on failure is the value of the constant MAP_FAILED; this was previously defined as –1.

Large File Summit extensions are added.

Issue 6

The *mmap*( ) function is marked as part of the Memory Mapped Files option.

The Open Group Corrigendum U028/6 is applied, changing (**void** *)–1 to MAP_FAILED.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to describe the use of MAP_FIXED.
- The DESCRIPTION is updated to describe the addition of an extra reference to the file associated with the file descriptor passed to *mmap*( ).
- The DESCRIPTION is updated to state that there may be implementation-defined limits on the number of memory regions that can be mapped.
- The DESCRIPTION is updated to describe constraints on the alignment and size of the *off* argument.
- The [EINVAL] and [EMFILE] error conditions are added.
- The [EOVERFLOW] error condition is added. This change is to support large files.

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- The DESCRIPTION is updated to describe the cases when MAP_PRIVATE and MAP_FIXED need not be supported.

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- Semantics for typed memory objects are added to the DESCRIPTION.
- New [ENOMEM] and [ENXIO] errors are added to the ERRORS section.
- The *posix_typed_mem_open*( ) function is added to the SEE ALSO section.

The normative text is updated to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/34 is applied, changing the margin code in the SYNOPSIS from MF|SHM to MC3 (notation for MF|SHM|TYM).

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/60 is applied, updating the DESCRIPTION and ERRORS sections to add the [EINVAL] error when *len* is zero.

Issue 7

Austin Group Interpretations 1003.1-2001 #078 and #079 are applied, clarifying page alignment requirements and adding a note about the state of synchronization objects becoming undefined when a shared region is unmapped.

Functionality relating to the Memory Protection and Memory Mapped Files options is moved to the Base.

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0229 [852] is applied.

Issue 8

Austin Group Defect 850 is applied, adding anonymous memory objects.

NAME

modf, modff, modfl — decompose a floating-point number

SYNOPSIS

```
#include <math.h>

double modf(double x, double *iptr);
float modff(float value, float *iptr);
long double modfl(long double value, long double *iptr);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall break the argument *x* into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part as a **double** (for the *modf*( ) function), a **float** (for the *modff*( ) function), or a **long double** (for the *modfl*( ) function), in the object pointed to by *iptr*.

RETURN VALUE

Upon successful completion, these functions shall return the signed fractional part of *x*.

MX The returned value shall be exact and shall be independent of the current rounding direction mode.

If *x* is NaN, a NaN shall be returned, and **iptr* shall be set to a NaN.

If *x* is ±Inf, ±0 shall be returned, and **iptr* shall be set to ±Inf.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *modf*( ) function computes the function result and **iptr* such that:

```
a = modf(x, iptr) ;
x == a+*iptr ;
```

allowing for the usual floating-point inaccuracies.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

frexp( ), *isnan*( ), *ldexp*( )

XBD **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *modff*( ) and *modfl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/35 is applied, correcting the code example in the APPLICATION USAGE section.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018 standard.

48633 **NAME**

48634 mprotect — set protection of memory mapping

48635 **SYNOPSIS**

48636 `#include <sys/mman.h>`

48637 `int mprotect(void *addr, size_t len, int prot);`

48638 **DESCRIPTION**

48639 The *mprotect*( ) function shall change the access protections to be that specified by *prot* for those
48640 whole pages containing any part of the address space of the process starting at address *addr* and
48641 continuing for *len* bytes. The parameter *prot* determines whether read, write, execute, or some
48642 combination of accesses are permitted to the data being mapped. The *prot* argument should be
48643 either PROT_NONE or the bitwise-inclusive OR of one or more of PROT_READ, PROT_WRITE,
48644 and PROT_EXEC.

48645 If an implementation cannot support the combination of access types specified by *prot*, the call to
48646 *mprotect*( ) shall fail.

48647 An implementation may permit accesses other than those specified by *prot*; however, no
48648 implementation shall permit a write to succeed where PROT_WRITE has not been set or shall
48649 permit any access where PROT_NONE alone has been set. Implementations shall support at
48650 least the following values of *prot*: PROT_NONE, PROT_READ, PROT_WRITE, and the bitwise-
48651 inclusive OR of PROT_READ and PROT_WRITE. If PROT_WRITE is specified, the application
48652 shall ensure that it has opened the mapped objects in the specified address range with write
48653 permission, unless MAP_PRIVATE was specified in the original mapping, regardless of whether
48654 the file descriptors used to map the objects have since been closed.

48655 The implementation may require that *addr* be a multiple of the page size as returned by
48656 *sysconf*( ).

48657 The behavior of this function is unspecified if the mapping was not established by a call to
48658 *mmap*( ).

48659 When *mprotect*( ) fails for reasons other than [EINVAL], the protections on some of the pages in
48660 the range [*addr*,*addr*+*len*) may have been changed.

48661 **RETURN VALUE**

48662 Upon successful completion, *mprotect*( ) shall return 0; otherwise, it shall return –1 and set *errno*
48663 to indicate the error.

48664 **ERRORS**

48665 The *mprotect*( ) function shall fail if:

48666 [EACCES] The *prot* argument specifies a protection that violates the access permission the
48667 process has to the underlying memory object.

48668 [EAGAIN] The *prot* argument specifies PROT_WRITE over a MAP_PRIVATE mapping
48669 and there are insufficient memory resources to reserve for locking the private
48670 page.

48671 [ENOMEM] Addresses in the range [*addr*,*addr*+*len*) are invalid for the address space of a
48672 process, or specify one or more pages which are not mapped.

48673 [ENOMEM] The *prot* argument specifies PROT_WRITE on a MAP_PRIVATE mapping, and
48674 it would require more space than the system is able to supply for locking the
48675 private pages, if required.

[ENOTSUP] The implementation does not support the combination of accesses requested in the *prot* argument.

The *mprotect*( ) function may fail if:

[EINVAL] The *addr* argument is not a multiple of the page size as returned by *sysconf*( ).

EXAMPLES

None.

APPLICATION USAGE

Most implementations require that *addr* is a multiple of the page size as returned by *sysconf*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mmap( ), *sysconf*( )

XBD **<sys/mman.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Aligned with *mprotect*( ) in the POSIX Realtime Extension as follows:

- The DESCRIPTION is largely reworded.
- [ENOTSUP] and a second form of [ENOMEM] are added as mandatory error conditions.
- [EAGAIN] is moved from the optional to the mandatory error conditions.

Issue 6

The *mprotect*( ) function is marked as part of the Memory Protection option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of the page size as returned by *sysconf*( ).
- The [EINVAL] error condition is added.

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

SD5-XSH-ERN-22 is applied, deleting erroneous APPLICATION USAGE.

Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment requirements.

The *mprotect*( ) function is moved from the Memory Protection option to the Base.

NAME

mq_close — close a message queue (**REALTIME**)

SYNOPSIS

MSG

```
#include <mqueue.h>

int mq_close(mqd_t mqdes);
```

DESCRIPTION

The *mq_close*( ) function shall remove the association between the message queue descriptor, *mqdes*, and its message queue. The results of using this message queue descriptor after successful return from this *mq_close*( ), and until the return of this message queue descriptor from a subsequent *mq_open*( ), are undefined.

If a message queue descriptor is implemented using a file descriptor, *mq_close*( ) shall close the file descriptor.

If the process has successfully attached a notification request to the message queue via this *mqdes*, this attachment shall be removed, and the message queue is available for another process to attach for notification.

RETURN VALUE

Upon successful completion, the *mq_close*( ) function shall return a value of zero; otherwise, the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *mq_close*( ) function shall fail if:

[EBADF] The *mqdes* argument is not a valid message queue descriptor.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_open( ), *mq_unlink*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( )

XBD **<mqueue.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *mq_close*( ) function is marked as part of the Message Passing option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

Issue 8

Austin Group Defect 368 is applied, adding a requirement that if a message queue descriptor is implemented using a file descriptor, *mq_close*( ) closes the file descriptor.

NAME

mq_getattr — get message queue attributes (**REALTIME**)

SYNOPSIS

MSG

```
#include <mqueue.h>

int mq_getattr(mqd_t mqdes, struct mq_attr *mqstat);
```

DESCRIPTION

The *mq_getattr*( ) function shall obtain status information and attributes of the message queue and the open message queue description associated with the message queue descriptor.

The *mqdes* argument specifies a message queue descriptor.

The results shall be returned in the **mq_attr** structure referenced by the *mqstat* argument.

Upon return, the following members shall have the values associated with the open message queue description as set when the message queue was opened and as modified by subsequent *mq_setattr*( ) calls: *mq_flags*.

The following attributes of the message queue shall be returned as set at message queue creation: *mq_maxmsg*, *mq_msgsize*.

Upon return, the following members within the **mq_attr** structure referenced by the *mqstat* argument shall be set to the current state of the message queue:

mq_curmsgs The number of messages currently on the queue.

RETURN VALUE

Upon successful completion, the *mq_getattr*( ) function shall return zero. Otherwise, the function shall return –1 and set *errno* to indicate the error.

ERRORS

The *mq_getattr*( ) function may fail if:

[EBADF] The *mqdes* argument is not a valid message queue descriptor.

EXAMPLES

See *mq_notify*( ).

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_notify( ), *mq_open*( ), *mq_send*( ), *mq_setattr*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( )

XBD **<mqueue.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *mq_getattr*( ) function is marked as part of the Message Passing option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

The *mq_timedsend*( ) function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/61 is applied, updating the ERRORS section to change the [EBADF] error from mandatory to optional.

NAME

mq_notify — notify process that a message is available (**REALTIME**)

SYNOPSIS

MSG

```
#include <mqueue.h>

int mq_notify(mqd_t mqdes, const struct sigevent *notification);
```

DESCRIPTION

If the argument *notification* is not NULL, this function shall register the calling process to be notified of message arrival at an empty message queue associated with the specified message queue descriptor, *mqdes*. The notification specified by the *notification* argument shall be sent to the process when the message queue transitions from empty to non-empty. At any time, only one process may be registered for notification by a message queue. If the calling process or any other process has already registered for notification of message arrival at the specified message queue, subsequent attempts to register for that message queue shall fail.

If *notification* is NULL and the process is currently registered for notification by the specified message queue, the existing registration shall be removed.

When the notification is sent to the registered process, its registration shall be removed. The message queue shall then be available for registration.

If a process has registered for notification of message arrival at a message queue and some thread is blocked in *mq_receive*( ) or *mq_timedreceive*( ) waiting to receive a message when a message arrives at the queue, the arriving message shall satisfy the appropriate *mq_receive*( ) or *mq_timedreceive*( ), respectively. The resulting behavior is as if the message queue remains empty, and no notification shall be sent.

RETURN VALUE

Upon successful completion, the *mq_notify*( ) function shall return a value of zero; otherwise, the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *mq_notify*( ) function shall fail if:

[EBADF] The *mqdes* argument is not a valid message queue descriptor.

[EBUSY] A process is already registered for notification by the message queue.

The *mq_notify*( ) function may fail if:

[EINVAL] The *notification* argument is NULL and the process is currently not registered.

EXAMPLES

The following program registers a notification request for the message queue named in its command-line argument. Notification is performed by creating a thread. The thread executes a function which reads one message from the queue and then terminates the process.

```
#include <pthread.h>
#include <signal.h>
#include <mqueue.h>
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

static void                     /* Thread start function */
tfunc(union sigval sv)
```



```
48842         {
48843             struct mq_attr attr;
48844             ssize_t nr;
48845             void *buf;
48846             mqd_t mqdes = *((mqd_t *) sv.sival_ptr);
48847             /* Determine maximum msg size; allocate buffer to receive msg */
48848             if (mq_getattr(mqdes, &attr) == -1) {
48849                 perror("mq_getattr");
48850                 exit(EXIT_FAILURE);
48851             }
48852             buf = malloc(attr.mq_msgsize);
48853             if (buf == NULL) {
48854                 perror("malloc");
48855                 exit(EXIT_FAILURE);
48856             }
48857             nr = mq_receive(mqdes, buf, attr.mq_msgsize, NULL);
48858             if (nr == -1) {
48859                 perror("mq_receive");
48860                 exit(EXIT_FAILURE);
48861             }
48862             printf("Read %ld bytes from message queue\n", (long) nr);
48863             free(buf);
48864             exit(EXIT_SUCCESS);         /* Terminate the process */
48865         }
48866         int
48867         main(int argc, char *argv[])
48868         {
48869             mqd_t mqdes;
48870             struct sigevent not;
48871             assert(argc == 2);
48872             mqdes = mq_open(argv[1], O_RDONLY);
48873             if (mqdes == (mqd_t) -1) {
48874                 perror("mq_open");
48875                 exit(EXIT_FAILURE);
48876             }
48877             not.sigev_notify = SIGEV_THREAD;
48878             not.sigev_notify_function = tfunc;
48879             not.sigev_notify_attributes = NULL;
48880             not.sigev_value.sival_ptr = &mqdes;   /* Arg. to thread func. */
48881             if (mq_notify(mqdes, &not) == -1) {
48882                 perror("mq_notify");
48883                 exit(EXIT_FAILURE);
48884             }
48885             pause();    /* Process will be terminated by thread function */
48886         }
```


48887 **APPLICATION USAGE**

48888 Since the **<mqueue.h>** header is only required to declare the **sigevent** structure tag as naming
48889 an incomplete structure type, in order to use *mq_notify*( ) and pass it a pointer to a **sigevent**
48890 structure, applications need to include **<signal.h>** so that **sigevent** will be fully defined.

48891 **RATIONALE**

48892 None.

48893 **FUTURE DIRECTIONS**

48894 None.

48895 **SEE ALSO**

48896 *mq_open*( ), *mq_send*( ), *mq_receive*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( )

48897 XBD **<mqueue.h>**

48898 **CHANGE HISTORY**

48899 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

48900 **Issue 6**

48901 The *mq_notify*( ) function is marked as part of the Message Passing option.

48902 The [ENOSYS] error condition has been removed as stubs need not be provided if an
48903 implementation does not support the Message Passing option.

48904 The *mq_timedsend*( ) function is added to the SEE ALSO section for alignment with IEEE Std
48905 1003.1d-1999.

48906 **Issue 7**

48907 SD5-XSH-ERN-38 is applied, adding the *mq_timedreceive*( ) function to the DESCRIPTION.

48908 Austin Group Interpretation 1003.1-2001 #032 is applied, adding the [EINVAL] error.

48909 An example is added.

48910 **Issue 8**

48911 Austin Group Defect 1282 is applied, changing the EXAMPLES and APPLICATION USAGE
48912 sections.

NAME

mq_open — open a message queue (**REALTIME**)

SYNOPSIS

MSG

```
#include <mqueue.h>

mqd_t mq_open(const char *name, int oflag, ...);
```

DESCRIPTION

The *mq_open*( ) function shall establish the connection between a process and a message queue with a message queue descriptor. It shall create an open message queue description that refers to the message queue, and a message queue descriptor that refers to that open message queue description. The message queue descriptor is used by other functions to refer to that message queue. The *name* argument points to a string naming a message queue. It is unspecified whether the name appears in the file system and is visible to other functions that take pathnames as arguments. The *name* argument conforms to the construction rules for a pathname, except that the interpretation of <slash> characters other than the leading <slash> character in *name* is implementation-defined, and that the length limits for the *name* argument are implementation-defined and need not be the same as the pathname limits {PATH_MAX} and {NAME_MAX}. If *name* begins with the <slash> character, then processes calling *mq_open*( ) with the same value of *name* shall refer to the same message queue object, as long as that name has not been removed. If *name* does not begin with the <slash> character, the effect is implementation-defined. If the *name* argument is not the name of an existing message queue and creation is not requested, *mq_open*( ) shall fail and return an error.

A message queue descriptor may be implemented using a file descriptor, in which case applications can open up to at least {OPEN_MAX} file and message queues.

The *oflag* argument requests the desired receive and/or send access to the message queue. The requested access permission to receive messages or send messages shall be granted if the calling process would be granted read or write access, respectively, to an equivalently protected file.

The value of *oflag* is the bitwise-inclusive OR of values from the following list. Applications shall specify exactly one of the first three values (access modes) below in the value of *oflag*:

O_RDONLY Open the message queue for receiving messages. The process can use the returned message queue descriptor with *mq_receive*( ), but not *mq_send*( ). A message queue may be open multiple times in the same or different processes for receiving messages.

O_WRONLY Open the queue for sending messages. The process can use the returned message queue descriptor with *mq_send*( ) but not *mq_receive*( ). A message queue may be open multiple times in the same or different processes for sending messages.

O_RDWR Open the queue for both receiving and sending messages. The process can use any of the functions allowed for O_RDONLY and O_WRONLY. A message queue may be open multiple times in the same or different processes for sending messages.

Any combination of the remaining flags may be specified in the value of *oflag*:

O_CREAT Create a message queue. It requires two additional arguments: *mode*, which shall be of type **mode_t**, and *attr*, which shall be a pointer to an **mq_attr** structure. If the pathname *name* has already been used to create a message queue that still exists, then this flag shall have no effect, except as noted under O_EXCL. Otherwise, a message queue shall be created without any messages

in it. The user ID of the message queue shall be set to the effective user ID of the process. The group ID of the message queue shall be set to the effective group ID of the process; however, if the *name* argument is visible in the file system, the group ID may be set to the group ID of the containing directory. When bits in *mode* other than the file permission bits are specified, the effect is unspecified. If *attr* is NULL, the message queue shall be created with implementation-defined default message queue attributes. If *attr* is non-NULL and the calling process has appropriate privileges on *name*, the message queue *mq_maxmsg* and *mq_msgsize* attributes shall be set to the values of the corresponding members in the **mq_attr** structure referred to by *attr*. The values of the *mq_flags* and *mq_curmsgs* members of the **mq_attr** structure shall be ignored. If *attr* is non-NULL, but the calling process does not have appropriate privileges on *name*, the *mq_open*( ) function shall fail and return an error without creating the message queue.

O_EXCL If O_EXCL and O_CREAT are set, *mq_open*( ) shall fail if the message queue *name* exists. The check for the existence of the message queue and the creation of the message queue if it does not exist shall be atomic with respect to other threads executing *mq_open*( ) naming the same *name* with O_EXCL and O_CREAT set. If O_EXCL is set and O_CREAT is not set, the result is undefined.

O_NONBLOCK Determines whether an *mq_send*( ) or *mq_receive*( ) waits for resources or messages that are not currently available, or fails with *errno* set to [EAGAIN]; see *mq_send*( ) and *mq_receive*( ) for details.

The *mq_open*( ) function does not add or remove messages from the queue.

RETURN VALUE

Upon successful completion, the function shall return a message queue descriptor; otherwise, the function shall return (**mqd_t**)−1 and set *errno* to indicate the error.

ERRORS

The *mq_open*( ) function shall fail if:

[EACCES] The message queue exists and the permissions specified by *oflag* are denied, or the message queue does not exist and permission to create the message queue is denied.

[EEXIST] O_CREAT and O_EXCL are set and the named message queue already exists.

[EINTR] The *mq_open*( ) function was interrupted by a signal.

[EINVAL] The *mq_open*( ) function is not supported for the given name.

[EINVAL] O_CREAT was specified in *oflag*, the value of *attr* is not NULL, and either *mq_maxmsg* or *mq_msgsize* was less than or equal to zero.

[EMFILE] Too many message queue descriptors or file descriptors are currently in use by this process.

[ENFILE] Too many message queue descriptors or file descriptors are currently open in the system.

[ENOENT] O_CREAT is not set and the named message queue does not exist.

[ENOSPC] There is insufficient space for the creation of the new message queue.

If any of the following conditions occur, the *mq_open*( ) function may return (**mqd_t**)–1 and set *errno* to the corresponding value.

[ENAMETOOLONG]
The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems that do not support the XSI option XSI or exceeds {_XOPEN_PATH_MAX} on XSI systems, or has a pathname component that is longer than {_POSIX_NAME_MAX} on systems that do not support the XSI option XSI or longer than {_XOPEN_NAME_MAX} on XSI systems.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

A future version might require the *mq_open*( ) and *mq_unlink*( ) functions to have semantics similar to normal file system operations.

SEE ALSO

mq_close( ), *mq_getattr*( ), *mq_receive*( ), *mq_send*( ), *mq_setattr*( ), *mq_unlink*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( )

XBD **<mqueue.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *mq_open*( ) function is marked as part of the Message Passing option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

The *mq_timedreceive*( ) and *mq_timedsend*( ) functions are added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

The DESCRIPTION of O_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/62 is applied, updating the description of the permission bits in the DESCRIPTION. The change is made for consistency with the *shm_open*( ) and *sem_open*( ) functions.

Issue 7

Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and changing [ENAMETOOLONG] from a ``shall fail'' to a ``may fail'' error.

Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the user ID and group ID of the message queue.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0394 [259] is applied.

Issue 8

Austin Group Defect 368 is applied, changing the [ENFILE] error.

NAME

mq_receive, mq_timedreceive — receive a message from a message queue (**REALTIME**)

SYNOPSIS

MSG

```
#include <mqueue.h>

ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
    unsigned *msg_prio);
ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
    size_t msg_len, unsigned *restrict msg_prio,
    const struct timespec *restrict abstime);
```

DESCRIPTION

The *mq_receive*( ) function shall receive the oldest of the highest priority message(s) from the message queue specified by *mqdes*. If the size of the buffer in bytes, specified by the *msg_len* argument, is less than the *mq_msgsize* attribute of the message queue, the function shall fail and return an error. Otherwise, the selected message shall be removed from the queue and copied to the buffer pointed to by the *msg_ptr* argument.

If the value of *msg_len* is greater than {SSIZE_MAX}, the result is implementation-defined.

If the argument *msg_prio* is not NULL, the priority of the selected message shall be stored in the location referenced by *msg_prio*.

If the specified message queue is empty and O_NONBLOCK is not set in the message queue description associated with *mqdes*, *mq_receive*( ) shall block until a message is enqueued on the message queue or until *mq_receive*( ) is interrupted by a signal. If more than one thread is waiting to receive a message when a message arrives at an empty queue and the Priority Scheduling option is supported, then the thread of highest priority that has been waiting the longest shall be selected to receive the message. Otherwise, it is unspecified which waiting thread receives the message. If the specified message queue is empty and O_NONBLOCK is set in the message queue description associated with *mqdes*, no message shall be removed from the queue, and *mq_receive*( ) shall return an error.

The *mq_timedreceive*( ) function shall receive the oldest of the highest priority messages from the message queue specified by *mqdes* as described for the *mq_receive*( ) function. However, if O_NONBLOCK was not specified when the message queue was opened via the *mq_open*( ) function, and no message exists on the queue to satisfy the receive, the wait for such a message shall be terminated when the specified timeout expires. If O_NONBLOCK is set, this function is equivalent to *mq_receive*( ).

The timeout expires when the absolute time specified by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the call.

The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall be the resolution of the clock on which it is based.

Under no circumstance shall the operation fail with a timeout if a message can be removed from the message queue immediately. The validity of the *abstime* parameter need not be checked if a message can be removed from the message queue immediately.

RETURN VALUE

Upon successful completion, the *mq_receive*( ) and *mq_timedreceive*( ) functions shall return the length of the selected message in bytes and the message shall be removed from the queue. Otherwise, no message shall be removed from the queue, the functions shall return a value of –1, and set *errno* to indicate the error.

ERRORS

These functions shall fail if:

[EAGAIN] O_NONBLOCK was set in the message description associated with *mqdes*, and the specified message queue is empty.

[EBADF] The *mqdes* argument is not a valid message queue descriptor open for reading.

[EMSGSIZE] The specified message buffer size, *msg_len*, is less than the message size attribute of the message queue.

[EINTR] The *mq_receive*( ) or *mq_timedreceive*( ) operation was interrupted by a signal.

[EINVAL] The process or thread would have blocked, and the *abstime* parameter specified a nanoseconds field value less than zero or greater than or equal to 1 000 million.

[ETIMEDOUT] The O_NONBLOCK flag was not set when the message queue was opened, but no message arrived on the queue before the specified timeout expired.

These functions may fail if:

[EBADMSG] The implementation has detected a data corruption problem with the message.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_open( ), *mq_send*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( ), *time*( )

XBD **<mqueue.h>**, **<time.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *mq_receive*( ) function is marked as part of the Message Passing option.

The Open Group Corrigendum U021/4 is applied. The DESCRIPTION is changed to refer to *msg_len* rather than *maxsize*.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In this function it is possible for the return value to exceed the range of the type **ssize_t** (since **size_t** has a larger range of positive values than **ssize_t**). A sentence restricting the size of the **size_t** object is added to the description to resolve this conflict.

The *mq_timedreceive*( ) function is added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *mq_timedreceive*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE PASC Interpretation 1003.1 #109 is applied, correcting the return type for *mq_timedreceive*( ) from **int** to **ssize_t**.

Issue 7

The *mq_timedreceive*( ) function is moved from the Timeouts option to the Base.

Functionality relating to the Timers option is moved to the Base.

Issue 8

Austin Group Defect 592 is applied, removing text relating to **<time.h>** from the SYNOPSIS and DESCRIPTION sections.

NAME

mq_send, mq_timedsend — send a message to a message queue (**REALTIME**)

SYNOPSIS

MSG

```
#include <mqueue.h>

int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
    unsigned msg_prio);
int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
    unsigned msg_prio, const struct timespec *abstime);
```

DESCRIPTION

The *mq_send*( ) function shall add the message pointed to by the argument *msg_ptr* to the message queue specified by *mqdes*. The *msg_len* argument specifies the length of the message, in bytes, pointed to by *msg_ptr*. The value of *msg_len* shall be less than or equal to the *mq_msgsize* attribute of the message queue, or *mq_send*( ) shall fail.

If the specified message queue is not full, *mq_send*( ) shall behave as if the message is inserted into the message queue at the position indicated by the *msg_prio* argument. A message with a larger numeric value of *msg_prio* shall be inserted before messages with lower values of *msg_prio*. A message shall be inserted after other messages in the queue, if any, with equal *msg_prio*. The value of *msg_prio* shall be less than {MQ_PRIO_MAX}.

If the specified message queue is full and O_NONBLOCK is not set in the message queue description associated with *mqdes*, *mq_send*( ) shall block until space becomes available to enqueue the message, or until *mq_send*( ) is interrupted by a signal. If more than one thread is waiting to send when space becomes available in the message queue and the Priority Scheduling option is supported, then the thread of the highest priority that has been waiting the longest shall be unblocked to send its message. Otherwise, it is unspecified which waiting thread is unblocked. If the specified message queue is full and O_NONBLOCK is set in the message queue description associated with *mqdes*, the message shall not be queued and *mq_send*( ) shall return an error.

The *mq_timedsend*( ) function shall add a message to the message queue specified by *mqdes* in the manner defined for the *mq_send*( ) function. However, if the specified message queue is full and O_NONBLOCK is not set in the message queue description associated with *mqdes*, the wait for sufficient room in the queue shall be terminated when the specified timeout expires. If O_NONBLOCK is set in the message queue description, this function shall be equivalent to *mq_send*( ).

The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the call.

The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall be the resolution of the clock on which it is based.

Under no circumstance shall the operation fail with a timeout if there is sufficient room in the queue to add the message immediately. The validity of the *abstime* parameter need not be checked when there is sufficient room in the queue.

RETURN VALUE

Upon successful completion, the *mq_send*( ) and *mq_timedsend*( ) functions shall return a value of zero. Otherwise, no message shall be enqueued, the functions shall return –1, and *errno* shall be set to indicate the error.

ERRORS

The *mq_send*( ) and *mq_timedsend*( ) functions shall fail if:

[EAGAIN] The O_NONBLOCK flag is set in the message queue description associated with *mqdes*, and the specified message queue is full.

[EBADF] The *mqdes* argument is not a valid message queue descriptor open for writing.

[EINTR] A signal interrupted the call to *mq_send*( ) or *mq_timedsend*( ).

[EINVAL] The value of *msg_prio* was outside the valid range.

[EINVAL] The process or thread would have blocked, and the *abstime* parameter specified a nanoseconds field value less than zero or greater than or equal to 1 000 million.

[EMSGSIZE] The specified message length, *msg_len*, exceeds the message size attribute of the message queue.

[ETIMEDOUT] The O_NONBLOCK flag was not set when the message queue was opened, but the timeout expired before the message could be added to the queue.

EXAMPLES

None.

APPLICATION USAGE

The value of the symbol {MQ_PRIO_MAX} limits the number of priority levels supported by the application. Message priorities range from 0 to {MQ_PRIO_MAX}−1.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_open( ), *mq_receive*( ), *mq_setattr*( ), *time*( )

XBD **<mqueue.h>**, **<time.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *mq_send*( ) function is marked as part of the Message Passing option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

The *mq_timedsend*( ) function is added for alignment with IEEE Std 1003.1d-1999.

Issue 7

The *mq_timedsend*( ) function is moved from the Timeouts option to the Base.

Functionality relating to the Timers option is moved to the Base.

Issue 8

Austin Group Defect 592 is applied, removing text relating to **<time.h>** from the SYNOPSIS and DESCRIPTION sections.

NAME

mq_setattr — set message queue attributes (**REALTIME**)

SYNOPSIS

MSG

```
#include <mqueue.h>

int mq_setattr(mqd_t mqdes, const struct mq_attr *restrict mqstat,
    struct mq_attr *restrict omqstat);
```

DESCRIPTION

The *mq_setattr*( ) function shall set attributes associated with the open message queue description referenced by the message queue descriptor specified by *mqdes*.

The message queue attributes corresponding to the following members defined in the **mq_attr** structure shall be set to the specified values upon successful completion of *mq_setattr*( ):

mq_flags The value of this member is the bitwise-logical OR of zero or more of O_NONBLOCK and any implementation-defined flags.

The values of the *mq_maxmsg*, *mq_msgsize*, and *mq_curmsgs* members of the **mq_attr** structure shall be ignored by *mq_setattr*( ).

If *omqstat* is non-NULL, the *mq_setattr*( ) function shall store, in the location referenced by *omqstat*, the previous message queue attributes and the current queue status. These values shall be the same as would be returned by a call to *mq_getattr*( ) at that point.

RETURN VALUE

Upon successful completion, the function shall return a value of zero and the attributes of the message queue shall have been changed as specified.

Otherwise, the message queue attributes shall be unchanged, and the function shall return a value of –1 and set *errno* to indicate the error.

ERRORS

The *mq_setattr*( ) function shall fail if:

[EBADF] The *mqdes* argument is not a valid message queue descriptor.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_open( ), *mq_send*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( )

XBD **<mqueue.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *mq_setattr*( ) function is marked as part of the Message Passing option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

The *mq_timedsend*( ) function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *mq_setattr*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

NAME

mq_timedreceive — receive a message from a message queue (**ADVANCED REALTIME**)

SYNOPSIS

MSG

```
#include <mqueue.h>

ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
    size_t msg_len, unsigned *restrict msg_prio,
    const struct timespec *restrict abstime);
```

DESCRIPTION

Refer to *mq_receive( )*.

NAME

mq_timedsend — send a message to a message queue (**ADVANCED REALTIME**)

SYNOPSIS

MSG

```
#include <mqueue.h>

int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
    unsigned msg_prio, const struct timespec *abstime);
```

DESCRIPTION

Refer to *mq_send( )*.

NAME

mq_unlink — remove a message queue (**REALTIME**)

SYNOPSIS

MSG

```
#include <mqueue.h>

int mq_unlink(const char *name);
```

DESCRIPTION

The *mq_unlink*( ) function shall remove the message queue named by the string name. If one or more processes have the message queue open when *mq_unlink*( ) is called, destruction of the message queue shall be postponed until all references to the message queue have been closed. However, the *mq_unlink*( ) call need not block until all references have been closed; it may return immediately.

After a successful call to *mq_unlink*( ), reuse of the name shall subsequently cause *mq_open*( ) to behave as if no message queue of this name exists (that is, *mq_open*( ) shall fail if O_CREAT is not set, or shall create a new message queue if O_CREAT is set).

RETURN VALUE

Upon successful completion, the function shall return a value of zero. Otherwise, the named message queue shall be unchanged by this function call, and the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *mq_unlink*( ) function shall fail if:

[EACCES] Permission is denied to unlink the named message queue.

[EINTR] The call to *mq_unlink*( ) blocked waiting for all references to the named message queue to be closed and a signal interrupted the call.

[ENOENT] The named message queue does not exist.

The *mq_unlink*( ) function may fail if:

[ENAMETOOLONG]

The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems XSI that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI systems, or has a pathname component that is longer than XSI {_POSIX_NAME_MAX} on systems that do not support the XSI option or longer than {_XOPEN_NAME_MAX} on XSI systems. A call to *mq_unlink*( ) with a *name* argument that contains the same message queue name as was previously used in a successful *mq_open*( ) call shall not give an [ENAMETOOLONG] error.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

A future version might require the *mq_open*( ) and *mq_unlink*( ) functions to have semantics similar to normal file system operations.

SEE ALSO

mq_close( ), *mq_open*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( )

XBD **<mqueue.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *mq_unlink*( ) function is marked as part of the Message Passing option.

The Open Group Corrigendum U021/5 is applied, clarifying that upon unsuccessful completion, the named message queue is unchanged by this function.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

Issue 7

Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a ``shall fail'' to a ``may fail'' error .

Austin Group Interpretation 1003.1-2001 #141 is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0230 [504] is applied.

NAME

mrand48 — generate uniformly distributed pseudo-random signed long integers

SYNOPSIS

XSI
```
#include <stdlib.h>

long mrand48(void);
```

DESCRIPTION

Refer to *drand48( )*.

NAME

msgctl — XSI message control operations

SYNOPSIS

XSI
```
#include <sys/msg.h>

int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

DESCRIPTION

The *msgctl*( ) function operates on XSI message queues (see XBD Section 3.206, on page 61). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 527).

The *msgctl*( ) function shall provide message control operations as specified by *cmd*. The following values for *cmd*, and the message control operations they specify, are:

IPC_STAT
: Place the current value of each member of the **msqid_ds** data structure associated with *msqid* into the structure pointed to by *buf*. The contents of this structure are defined in **<sys/msg.h>**.

IPC_SET
: Set the value of the following members of the **msqid_ds** data structure associated with *msqid* to the corresponding value found in the structure pointed to by *buf*:

```
msg_perm.uid
msg_perm.gid
msg_perm.mode
msg_qbytes
```

Also, the *msg_ctime* timestamp shall be set to the current time, as described in Section 2.7.1 (on page 526).

IPC_SET can only be executed by a process with appropriate privileges or that has an effective user ID equal to the value of **msg_perm.cuid** or **msg_perm.uid** in the **msqid_ds** data structure associated with *msqid*. Only a process with appropriate privileges can raise the value of **msg_qbytes**.

IPC_RMID
: Remove the message queue identifier specified by *msqid* from the system and destroy the message queue and **msqid_ds** data structure associated with it. IPC_RMD can only be executed by a process with appropriate privileges or one that has an effective user ID equal to the value of **msg_perm.cuid** or **msg_perm.uid** in the **msqid_ds** data structure associated with *msqid*.

RETURN VALUE

Upon successful completion, *msgctl*( ) shall return 0; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *msgctl*( ) function shall fail if:

[EACCES]
: The argument *cmd* is IPC_STAT and the calling process does not have read permission; see Section 2.7 (on page 526).

[EINVAL]
: The value of *msqid* is not a valid message queue identifier; or the value of *cmd* is not a valid command.

[EPERM]
: The argument *cmd* is IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of **msg_perm.cuid** or **msg_perm.uid** in the data

49401 structure associated with *msqid*.

49402 [EPERM] The argument *cmd* is IPC_SET, an attempt is being made to increase to the
49403 value of **msg_qbytes**, and the effective user ID of the calling process does not
49404 have appropriate privileges.

49405 **EXAMPLES**

49406 None.

49407 **APPLICATION USAGE**

49408 The POSIX Realtime Extension defines alternative interfaces for interprocess communication
49409 (IPC). Application developers who need to use IPC should design their applications so that
49410 modules using the IPC routines described in Section 2.7 (on page 526) can be easily modified to
49411 use the alternative interfaces.

49412 **RATIONALE**

49413 None.

49414 **FUTURE DIRECTIONS**

49415 None.

49416 **SEE ALSO**

49417 Section 2.7 (on page 526), Section 2.8 (on page 527), *mq_close*( ), *mq_getattr*( ), *mq_notify*( ),
49418 *mq_open*( ), *mq_receive*( ), *mq_send*( ), *mq_setattr*( ), *mq_unlink*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( )

49419 XBD Section 3.206 (on page 61), **<sys/msg.h>**

49420 **CHANGE HISTORY**

49421 First released in Issue 2. Derived from Issue 2 of the SVID.

49422 **Issue 5**

49423 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
49424 DIRECTIONS to a new APPLICATION USAGE section.

49425 **Issue 7**

49426 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0395 [345] is applied.

NAME

msgget — get the XSI message queue identifier

SYNOPSIS

XSI
```
#include <sys/msg.h>

int msgget(key_t key, int msgflg);
```

DESCRIPTION

The *msgget*( ) function operates on XSI message queues (see XBD Section 3.206, on page 61). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 527).

The *msgget*( ) function shall return the message queue identifier associated with the argument *key*.

A message queue identifier, associated message queue, and data structure (see **<sys/msg.h>**), shall be created for the argument *key* if one of the following is true:

- The argument *key* is equal to IPC_PRIVATE.
- The argument *key* does not already have a message queue identifier associated with it, and (*msgflg* & IPC_CREAT) is non-zero.

Upon creation, the data structure associated with the new message queue identifier shall be initialized as follows:

- **msg_perm.cuid**, **msg_perm.uid**, **msg_perm.cgid**, and **msg_perm.gid** shall be set to the effective user ID and effective group ID, respectively, of the calling process.
- The low-order 9 bits of **msg_perm.mode** shall be set to the low-order 9 bits of *msgflg*.
- **msg_qnum**, **msg_lspid**, **msg_lrpid**, **msg_stime**, and **msg_rtime** shall be set to 0.
- **msg_ctime** shall be set to the current time, as described in Section 2.7.1 (on page 526).
- **msg_qbytes** shall be set to the system limit.

RETURN VALUE

Upon successful completion, *msgget*( ) shall return a non-negative integer, namely a message queue identifier. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *msgget*( ) function shall fail if:

[EACCES] A message queue identifier exists for the argument *key*, but operation permission as specified by the low-order 9 bits of *msgflg* would not be granted; see Section 2.7 (on page 526).

[EEXIST] A message queue identifier exists for the argument *key* but ((*msgflg* & IPC_CREAT) && (*msgflg* & IPC_EXCL)) is non-zero.

[ENOENT] A message queue identifier does not exist for the argument *key* and (*msgflg* & IPC_CREAT) is 0.

[ENOSPC] A message queue identifier is to be created but the system-imposed limit on the maximum number of allowed message queue identifiers system-wide would be exceeded.

EXAMPLES

None.

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication (IPC). Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 526) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7 (on page 526), Section 2.8 (on page 527), *ftok*( ), *mq_close*( ), *mq_getattr*( ), *mq_notify*( ), *mq_open*( ), *mq_receive*( ), *mq_send*( ), *mq_setattr*( ), *mq_unlink*( ), *msgctl*( ), *msgrcv*( ), *msgsnd*( )

XBD Section 3.206 (on page 61), **<sys/msg.h>**

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0396 [345] and XSH/TC1-2008/0397 [344] are applied.

NAME

msgrcv — XSI message receive operation

SYNOPSIS

XSI
```
#include <sys/msg.h>

ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,
    int msgflg);
```

DESCRIPTION

The *msgrcv*( ) function operates on XSI message queues (see XBD Section 3.206, on page 61). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 527).

The *msgrcv*( ) function shall read a message from the queue associated with the message queue identifier specified by *msqid* and place it in the user-defined buffer pointed to by *msgp*.

The application shall ensure that the argument *msgp* points to a user-defined buffer that contains first a field of type **long** specifying the type of the message, and then a data portion that holds the data bytes of the message. The structure below is an example of what this user-defined buffer might look like:

```
struct mymsg {
    long    mtype;     /* Message type. */
    char    mtext[1];  /* Message text. */
}
```

The structure member *mtype* is the received message's type as specified by the sending process.

The structure member *mtext* is the text of the message.

The argument *msgsz* specifies the size in bytes of *mtext*. The received message shall be truncated to *msgsz* bytes if it is larger than *msgsz* and (*msgflg* & MSG_NOERROR) is non-zero. The truncated part of the message shall be lost and no indication of the truncation shall be given to the calling process.

If the value of *msgsz* is greater than {SSIZE_MAX}, the result is implementation-defined.

The argument *msgtyp* specifies the type of message requested as follows:

- If *msgtyp* is 0, the first message on the queue shall be received.
- If *msgtyp* is greater than 0, the first message of type *msgtyp* shall be received.
- If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the absolute value of *msgtyp* shall be received.

The argument *msgflg* specifies the action to be taken if a message of the desired type is not on the queue. These are as follows:

- If (*msgflg* & IPC_NOWAIT) is non-zero, the calling thread shall return immediately with a return value of –1 and *errno* set to [ENOMSG].
- If (*msgflg* & IPC_NOWAIT) is 0, the calling thread shall suspend execution until one of the following occurs:
  - — A message of the desired type is placed on the queue.
  - — The message queue identifier *msqid* is removed from the system; when this occurs, *errno* shall be set to [EIDRM] and –1 shall be returned.

— The calling thread receives a signal that is to be caught; in this case a message is not received and the calling thread resumes execution in the manner prescribed in *sigaction*( ).

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid*:

- **msg_qnum** shall be decremented by 1.
- **msg_lrpid** shall be set to the process ID of the calling process.
- **msg_rtime** shall be set to the current time, as described in Section 2.7.1 (on page 526).

RETURN VALUE

Upon successful completion, *msgrcv*( ) shall return a value equal to the number of bytes actually placed into the buffer *mtext*. Otherwise, no message shall be received, *msgrcv*( ) shall return –1, and *errno* shall be set to indicate the error.

ERRORS

The *msgrcv*( ) function shall fail if:

[E2BIG] The value of *mtext* is greater than *msgsz* and (*msgflg* & MSG_NOERROR) is 0.

[EACCES] Operation permission is denied to the calling process; see Section 2.7 (on page 526).

[EIDRM] The message queue identifier *msqid* is removed from the system.

[EINTR] The *msgrcv*( ) function was interrupted by a signal.

[EINVAL] *msqid* is not a valid message queue identifier.

[ENOMSG] The queue does not contain a message of the desired type and (*msgflg* & IPC_NOWAIT) is non-zero.

EXAMPLES

Receiving a Message

The following example receives the first message on the queue (based on the value of the *msgtyp* argument, 0). The queue is identified by the *msqid* argument (assuming that the value has previously been set). This call specifies that an error should be reported if no message is available, but not if the message is too large. The message size is calculated directly using the *sizeof* operator.

```
#include <sys/msg.h>
...
int result;
int msqid;
struct message {
    long type;
    char text[20];
} msg;
long msgtyp = 0;
...
result = msgrcv(msqid, (void *) &msg, sizeof(msg.text),
         msgtyp, MSG_NOERROR | IPC_NOWAIT);
```


APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication (IPC). Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 526) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7 (on page 526), Section 2.8 (on page 527), *mq_close*( ), *mq_getattr*( ), *mq_notify*( ), *mq_open*( ), *mq_receive*( ), *mq_send*( ), *mq_setattr*( ), *mq_unlink*( ), *msgctl*( ), *msgget*( ), *msgsnd*( ), *sigaction*( )

XBD Section 3.206 (on page 61), **<sys/msg.h>**

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

The type of the return value is changed from **int** to **ssize_t**, and a warning is added to the DESCRIPTION about values of *msgsz* larger the {SSIZE_MAX}.

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to the APPLICATION USAGE section.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0398 [345] and XSH/TC1-2008/0399 [421] are applied.

NAME

msgsnd — XSI message send operation

SYNOPSIS

XSI
```
#include <sys/msg.h>

int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

DESCRIPTION

The *msgsnd*( ) function operates on XSI message queues (see XBD Section 3.206, on page 61). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 527).

The *msgsnd*( ) function shall send a message to the queue associated with the message queue identifier specified by *msqid*.

The application shall ensure that the argument *msgp* points to a user-defined buffer that contains first a field of type **long** specifying the type of the message, and then a data portion that holds the data bytes of the message. The structure below is an example of what this user-defined buffer might look like:

```
struct mymsg {
    long   mtype;       /* Message type. */
    char   mtext[1];    /* Message text. */
}
```

The structure member *mtype* is a non-zero positive type **long** that can be used by the receiving process for message selection.

The structure member *mtext* is any text of length *msgsz* bytes. The argument *msgsz* can range from 0 to a system-imposed maximum.

The argument *msgflg* specifies the action to be taken if one or more of the following is true:

- The number of bytes already on the queue is equal to **msg_qbytes**; see **<sys/msg.h>**.
- The total number of messages on all queues system-wide is equal to the system-imposed limit.

These actions are as follows:

- If (*msgflg* & IPC_NOWAIT) is non-zero, the message shall not be sent and the calling thread shall return immediately.
- If (*msgflg* & IPC_NOWAIT) is 0, the calling thread shall suspend execution until one of the following occurs:
  - — The condition responsible for the suspension no longer exists, in which case the message is sent.
  - — The message queue identifier *msqid* is removed from the system; when this occurs, *errno* shall be set to [EIDRM] and −1 shall be returned.
  - — The calling thread receives a signal that is to be caught; in this case the message is not sent and the calling thread resumes execution in the manner prescribed in *sigaction*( ).

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid*; see **<sys/msg.h>**:

- **msg_qnum** shall be incremented by 1.
- **msg_lspid** shall be set to the process ID of the calling process.
- **msg_stime** shall be set to the current time, as described in Section 2.7.1 (on page 526).

RETURN VALUE

Upon successful completion, *msgsnd*( ) shall return 0; otherwise, no message shall be sent, *msgsnd*( ) shall return –1, and *errno* shall be set to indicate the error.

ERRORS

The *msgsnd*( ) function shall fail if:

[EACCES] Operation permission is denied to the calling process; see Section 2.7 (on page 526).

[EAGAIN] The message cannot be sent for one of the reasons cited above and (*msgflg* & IPC_NOWAIT) is non-zero.

[EIDRM] The message queue identifier *msqid* is removed from the system.

[EINTR] The *msgsnd*( ) function was interrupted by a signal.

[EINVAL] The value of *msqid* is not a valid message queue identifier, or the value of *mtype* is less than 1; or the value of *msgsz* is greater than the system-imposed limit.

EXAMPLES

Sending a Message

The following example sends a message to the queue identified by the *msqid* argument (assuming that value has previously been set). This call specifies that an error should be reported if no message is available. The message size is calculated directly using the *sizeof* operator.

```
#include <sys/msg.h>
...
int result;
int msqid;
struct message {
    long type;
    char text[20];
} msg;

msg.type = 1;
strcpy(msg.text, "This is message 1");
...
result = msgsnd(msqid, (void *) &msg, sizeof(msg.text), IPC_NOWAIT);
```

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication (IPC). Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 526) can be easily modified to use the alternative interfaces.

49677 **RATIONALE**

49678 None.

49679 **FUTURE DIRECTIONS**

49680 None.

49681 **SEE ALSO**

49682 Section 2.7 (on page 526), Section 2.8 (on page 527), *mq_close( )*, *mq_getattr( )*, *mq_notify( )*,
49683 *mq_open( )*, *mq_receive( )*, *mq_send( )*, *mq_setattr( )*, *mq_unlink( )*, *msgctl( )*, *msgget( )*, *msgrcv( )*,
49684 *sigaction( )*

49685 XBD Section 3.206 (on page 61), **<sys/msg.h>**

49686 **CHANGE HISTORY**

49687 First released in Issue 2. Derived from Issue 2 of the SVID.

49688 **Issue 5**

49689 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
49690 DIRECTIONS to a new APPLICATION USAGE section.

49691 **Issue 6**

49692 The normative text is updated to avoid use of the term ``must'' for application requirements.

49693 **Issue 7**

49694 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0400 [345] and XSH/TC1-2008/0401
49695 [359] are applied.

NAME

msync — synchronize memory with physical storage

SYNOPSIS

XSI|SIO
```
#include <sys/mman.h>

int msync(void *addr, size_t len, int flags);
```

DESCRIPTION

The *msync*( ) function shall write all modified data to permanent storage locations, if any, in those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes. If no such storage exists, *msync*( ) need not have any effect. If requested, the *msync*( ) function shall then invalidate cached copies of data.

The implementation may require that *addr* be a multiple of the page size as returned by *sysconf*( ).

For mappings to files, the *msync*( ) function shall ensure that all write operations are completed as defined for synchronized I/O data integrity completion. It is unspecified whether the implementation also writes out other file attributes. When the *msync*( ) function is called on MAP_PRIVATE mappings, any modified data shall not be written to the underlying object and shall not cause such data to be made visible to other processes. It is unspecified whether data in MAP_PRIVATE mappings has any permanent storage locations. The effect of *msync*( ) on an SHM anonymous memory object, shared memory object, or TYM typed memory object is unspecified. The behavior of this function is unspecified if the mapping was not established by a call to *mmap*( ).

The *flags* argument is constructed from the bitwise-inclusive OR of one or more of the following flags defined in the **<sys/mman.h>** header:

Symbolic Constant	Description
MS_ASYNC	Perform asynchronous writes.
MS_SYNC	Perform synchronous writes.
MS_INVALIDATE	Invalidate cached data.

When MS_ASYNC is specified, *msync*( ) shall return immediately once all the write operations are initiated or queued for servicing; when MS_SYNC is specified, *msync*( ) shall not return until all write operations are completed as defined for synchronized I/O data integrity completion. Either MS_ASYNC or MS_SYNC shall be specified, but not both.

When MS_INVALIDATE is specified, *msync*( ) shall invalidate all cached copies of mapped data that are inconsistent with the permanent storage locations such that subsequent references shall obtain data that was consistent with the permanent storage locations sometime between the call to *msync*( ) and the first subsequent memory reference to the data.

If *msync*( ) causes any write to a file, the file's last data modification and last file status change timestamps shall be marked for update.

RETURN VALUE

Upon successful completion, *msync*( ) shall return 0; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *msync*( ) function shall fail if:

[EBUSY] Some or all of the addresses in the range starting at *addr* and continuing for *len* bytes are locked, and MS_INVALIDATE is specified.

[EINVAL] The value of *flags* is invalid.

[ENOMEM] The addresses in the range starting at *addr* and continuing for *len* bytes are outside the range allowed for the address space of a process or specify one or more pages that are not mapped.

The *msync*( ) function may fail if:

[EINVAL] The value of *addr* is not a multiple of the page size as returned by *sysconf*( ).

EXAMPLES

None.

APPLICATION USAGE

The *msync*( ) function is only supported if the Synchronized Input and Output option is supported, and thus need not be available on all implementations.

The *msync*( ) function should be used by programs that require a memory object to be in a known state; for example, in building transaction facilities.

Normal system activity can cause pages to be written to disk. Therefore, there are no guarantees that *msync*( ) is the only control over when pages are or are not written to disk.

RATIONALE

The *msync*( ) function writes out data in a mapped region to the permanent storage for the underlying object. The call to *msync*( ) ensures data integrity of the file.

After the data is written out, any cached data may be invalidated if the MS_INVALIDATE flag was specified. This is useful on systems that do not support read/write consistency.

FUTURE DIRECTIONS

None.

SEE ALSO

mmap( ), *sysconf*( )

XBD **<sys/mman.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Aligned with *msync*( ) in the POSIX Realtime Extension as follows:

- The DESCRIPTION is extensively reworded.
- [EBUSY] and a new form of [EINVAL] are added as mandatory error conditions.

Issue 6

The *msync*( ) function is marked as part of the Memory Mapped Files and Synchronized Input and Output options.

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- The [EBUSY] mandatory error condition is added.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of the page size.
- The second [EINVAL] error condition is made mandatory.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding reference to typed memory objects.

Issue 7

Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment requirements.

SD5-XSH-ERN-110 is applied.

The *msync*( ) function is marked as part of the Synchronized Input and Output option or XSI option as the Memory Mapped Files is moved to the Base.

Changes are made related to support for finegrained timestamps.

Issue 8

Austin Group Defect 850 is applied, adding anonymous memory objects.

NAME

mtx_destroy, mtx_init — destroy and initialize a mutex

SYNOPSIS

```
#include <threads.h>

void mtx_destroy(mtx_t *mtx);
int  mtx_init(mtx_t *mtx, int type);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *mtx_destroy*( ) function shall release any resources used by the mutex pointed to by *mtx*. A destroyed mutex object can be reinitialized using *mtx_init*( ); the results of otherwise referencing the object after it has been destroyed are undefined. It shall be safe to destroy an initialized mutex that is unlocked. Attempting to destroy a locked mutex, or a mutex that another thread is attempting to lock, or a mutex that is being used in a *cnd_timedwait*( ) or *cnd_wait*( ) call by another thread, results in undefined behavior. The behavior is undefined if the value specified by the *mtx* argument to *mtx_destroy*( ) does not refer to an initialized mutex.

The *mtx_init*( ) function shall initialize a mutex object with properties indicated by *type*, whose valid values include:

`mtx_plain`	for a simple non-recursive mutex,
`mtx_timed`	for a non-recursive mutex that supports timeout,
`mtx_plain|mtx_recursive`	for a simple recursive mutex, or
`mtx_timed|mtx_recursive`	for a recursive mutex that supports timeout.

If the *mtx_init*( ) function succeeds, it shall set the mutex pointed to by *mtx* to a value that uniquely identifies the newly initialized mutex. Upon successful initialization, the state of the mutex shall become initialized and unlocked. Attempting to initialize an already initialized mutex results in undefined behavior.

CX See Section 2.9.9 (on page 548) for further requirements.

These functions shall not be affected if the calling thread executes a signal handler during the call.

RETURN VALUE

The *mtx_destroy*( ) function shall not return a value.

The *mtx_init*( ) function shall return `thrd_success` on success or `thrd_error` if the request could not be honored.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

A mutex can be destroyed immediately after it is unlocked. However, since attempting to destroy a locked mutex, or a mutex that another thread is attempting to lock, or a mutex that is being used in a *cnd_timedwait*( ) or *cnd_wait*( ) call by another thread results in undefined behavior, care must be taken to ensure that no other thread may be referencing the mutex.

RATIONALE

These functions are not affected by signal handlers for the reasons stated in XRAT Section B.2.3 (on page 3742).

FUTURE DIRECTIONS

None.

SEE ALSO

cnd_timedwait( ), *mtx_lock*( )

XBD **<threads.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

mtx_lock, mtx_timedlock, mtx_trylock, mtx_unlock — lock and unlock a mutex

SYNOPSIS

```
#include <threads.h>

int mtx_lock(mtx_t *mtx);
int mtx_timedlock(mtx_t *restrict mtx,
    const struct timespec *restrict ts);
int mtx_trylock(mtx_t *mtx);
int mtx_unlock(mtx_t *mtx);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *mtx_lock*( ) function shall block until it locks the mutex pointed to by *mtx*. If the mutex is non-recursive, the application shall ensure that it is not already locked by the calling thread.

The *mtx_timedlock*( ) function shall block until it locks the mutex pointed to by *mtx* or until after the TIME_UTC-based calendar time pointed to by *ts*. The application shall ensure that the CX specified mutex supports timeout. Under no circumstance shall the function fail with a timeout if the mutex can be locked immediately. The validity of the *ts* parameter need not be checked if the mutex can be locked immediately.

The *mtx_trylock*( ) function shall endeavor to lock the mutex pointed to by *mtx*. If the mutex is already locked (by any thread, including the current thread), the function shall return without blocking. If the mutex is recursive and the mutex is currently owned by the calling thread, the mutex lock count (see below) shall be incremented by one and the *mtx_trylock*( ) function shall immediately return success.

CX These functions shall not be affected if the calling thread executes a signal handler during the call; if a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the thread shall resume waiting for the mutex as if it was not interrupted.

If a call to *mtx_lock*( ), *mtx_timedlock*( ) or *mtx_trylock*( ) locks the mutex, prior calls to *mtx_unlock*( ) on the same mutex shall synchronize with this lock operation.

The *mtx_unlock*( ) function shall unlock the mutex pointed to by *mtx*. The application shall CX ensure that the mutex pointed to by *mtx* is locked by the calling thread. If there are threads blocked on the mutex object referenced by *mtx* when *mtx_unlock*( ) is called, resulting in the mutex becoming available, the scheduling policy shall determine which thread shall acquire the mutex.

A recursive mutex shall maintain the concept of a lock count. When a thread successfully acquires a mutex for the first time, the lock count shall be set to one. Every time a thread relocks this mutex, the lock count shall be incremented by one. Each time the thread unlocks the mutex, the lock count shall be decremented by one. When the lock count reaches zero, the mutex shall become available for other threads to acquire.

For purposes of determining the existence of a data race, mutex lock and unlock operations on mutexes of type **mtx_t** behave as atomic operations. All lock and unlock operations on a particular mutex occur in some particular total order.

If *mtx* does not refer to an initialized mutex object, the behavior of these functions is undefined.

RETURN VALUE

The *mtx_lock*( ) and *mtx_unlock*( ) functions shall return `thrd_success` on success, or `thrd_error` if the request could not be honored.

The *mtx_timedlock*( ) function shall return `thrd_success` on success, or `thrd_timedout` if the time specified was reached without acquiring the requested resource, or `thrd_error` if the request could not be honored.

The *mtx_trylock*( ) function shall return `thrd_success` on success, or `thrd_busy` if the resource requested is already in use, or `thrd_error` if the request could not be honored. The *mtx_trylock*( ) function can spuriously fail to lock an unused resource, in which case it shall return `thrd_busy`.

ERRORS

See RETURN VALUE.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

These functions are not affected by signal handlers for the reasons stated in XRAT Section B.2.3 (on page 3742).

Since **<pthread.h>** has no equivalent of the `mtx_timed` mutex property, if the **<threads.h>** interfaces are implemented as a thin wrapper around **<pthread.h>** interfaces (meaning **mtx_t** and **pthread_mutex_t** are the same type), all mutexes support timeout and *mtx_timedlock*( ) will not fail for a mutex that was not initialized with `mtx_timed`. Alternatively, implementations can use a less thin wrapper where **mtx_t** contains additional properties that are not held in **pthread_mutex_t** in order to be able to return a failure indication from *mtx_timedlock*( ) calls where the mutex was not initialized with `mtx_timed`.

FUTURE DIRECTIONS

None.

SEE ALSO

mtx_destroy( ), *timespec_get*( )

XBD **<threads.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

NAME

munlock — unlock a range of process address space

SYNOPSIS

MLR

```
#include <sys/mman.h>

int munlock(const void *addr, size_t len);
```

DESCRIPTION

Refer to *mlock*( ).

NAME

munlockall — unlock the address space of a process

SYNOPSIS

ML

```
#include <sys/mman.h>

int munlockall(void);
```

DESCRIPTION

Refer to *mlockall( )*.

NAME

munmap — unmap pages of memory

SYNOPSIS

```
#include <sys/mman.h>

int munmap(void *addr, size_t len);
```

DESCRIPTION

The *munmap*( ) function shall remove any mappings for those entire pages containing any part of the address space of the process starting at *addr* and continuing for *len* bytes. Further references to these pages shall result in the generation of a SIGSEGV signal to the process. If there are no mappings in the specified address range, then *munmap*( ) has no effect.

The implementation may require that *addr* be a multiple of the page size as returned by *sysconf*( ).

If a mapping to be removed was private, any modifications made in this address range shall be discarded.

ML|MLR Any memory locks (see *mlock*( ) and *mlockall*( )) associated with this address range shall be removed, as if by an appropriate call to *munlock*( ).

TYM If a mapping removed from a typed memory object causes the corresponding address range of the memory pool to be inaccessible by any process in the system except through allocatable mappings (that is, mappings of typed memory objects opened with the POSIX_TYPED_MEM_MAP_ALLOCATABLE flag), then that range of the memory pool shall become deallocated and may become available to satisfy future typed memory allocation requests.

A mapping removed from a typed memory object opened with the POSIX_TYPED_MEM_MAP_ALLOCATABLE flag shall not affect in any way the availability of that typed memory for allocation.

The behavior of this function is unspecified if the mapping was not established by a call to *mmap*( ).

RETURN VALUE

Upon successful completion, *munmap*( ) shall return 0; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *munmap*( ) function shall fail if:

[EINVAL] Addresses in the range [*addr*,*addr*+*len*) are outside the valid range for the address space of a process.

[EINVAL] The *len* argument is 0.

The *munmap*( ) function may fail if:

[EINVAL] The *addr* argument is not a multiple of the page size as returned by *sysconf*( ).

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *munmap*( ) function corresponds to SVR4, just as the *mmap*( ) function does.

It is possible that an application has applied process memory locking to a region that contains shared memory. If this has occurred, the *munmap*( ) call ignores those locks and, if necessary, causes those locks to be removed.

Most implementations require that *addr* is a multiple of the page size as returned by *sysconf*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

mlock( ), *mlockall*( ), *mmap*( ), *posix_typed_mem_open*( ), *sysconf*( )

XBD **<sys/mman.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Aligned with *munmap*( ) in the POSIX Realtime Extension as follows:

- The DESCRIPTION is extensively reworded.
- The SIGBUS error is no longer permitted to be generated.

Issue 6

The *munmap*( ) function is marked as part of the Memory Mapped Files and Shared Memory Objects option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of the page size.
- The [EINVAL] error conditions are added.

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- Semantics for typed memory objects are added to the DESCRIPTION.
- The *posix_typed_mem_open*( ) function is added to the SEE ALSO section.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/36 is applied, changing the margin code in the SYNOPSIS from MF|SHM to MC3 (notation for MF|SHM|TYM).

Issue 7

Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment requirements.

The *munmap*( ) function is moved from the Memory Mapped Files option to the Base.

NAME

nan, nanf, nanl — return quiet NaN

SYNOPSIS

```
#include <math.h>

double nan(const char *tagp);
float nanf(const char *tagp);
long double nanl(const char *tagp);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The function call *nan*("*n-char-sequence*") shall be equivalent to:

```
strtod("NAN(n-char-sequence)", (char **) NULL);
```

The function call *nan*("") shall be equivalent to:

```
strtod("NAN()", (char **) NULL)
```

If *tagp* does not point to an *n*-**char** sequence or an empty string, the function call shall be equivalent to:

```
strtod("NAN", (char **) NULL)
```

Function calls to *nanf*() and *nanl*() are equivalent to the corresponding function calls to *strtof*() and *strtold*().

RETURN VALUE

These functions shall return a quiet NaN, if available, with content indicated through *tagp*.

MX The returned value shall be exact and shall be independent of the current rounding direction mode.

If the implementation does not support quiet NaNs, these functions shall return zero.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strtod()

XBD **<math.h>**

50051 **CHANGE HISTORY**

50052 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

50053 **Issue 8**

50054 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018
50055 standard.

NAME

nanosleep — high resolution sleep

SYNOPSIS

CX
```
#include <time.h>

int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);
```

DESCRIPTION

The *nanosleep*( ) function shall cause the current thread to be suspended from execution until either the time interval specified by the *rqtp* argument has elapsed or a signal is delivered to the calling thread, and its action is to invoke a signal-catching function or to terminate the process. The suspension time may be longer than requested because the argument value is rounded up to an integer multiple of the sleep resolution or because of the scheduling of other activity by the system. But, except for the case of being interrupted by a signal, the suspension time shall not be less than the time specified by *rqtp*, as measured by the system clock CLOCK_REALTIME.

The use of the *nanosleep*( ) function has no effect on the action or blockage of any signal.

RETURN VALUE

If the *nanosleep*( ) function returns because the requested time has elapsed, its return value shall be zero.

If the *nanosleep*( ) function returns because it has been interrupted by a signal, it shall return a value of −1 and set *errno* to indicate the interruption. If the *rmtp* argument is non-NULL, the **timespec** structure referenced by it is updated to contain the amount of time remaining in the interval (the requested time minus the time actually slept). The *rqtp* and *rmtp* arguments can point to the same object. If the *rmtp* argument is NULL, the remaining time is not returned.

If *nanosleep*( ) fails, it shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *nanosleep*( ) function shall fail if:

[EINTR] The *nanosleep*( ) function was interrupted by a signal.

[EINVAL] The *rqtp* argument specified a nanosecond value less than zero or greater than or equal to 1 000 million.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

It is common to suspend execution of a thread for an interval in order to poll the status of a non-interrupting function. A large number of actual needs can be met with a simple extension to *sleep*( ) that provides finer resolution.

In the POSIX.1-1990 standard and SVR4, it is possible to implement such a routine, but the frequency of wakeup is limited by the resolution of the *alarm*( ) and *sleep*( ) functions. In 4.3 BSD, it is possible to write such a routine using no static storage and reserving no system facilities. Although it is possible to write a function with similar functionality to *sleep*( ) using the remainder of the *timer_**( ) functions, such a function requires the use of signals and the reservation of some signal number. This volume of POSIX.1-2024 requires that *nanosleep*( ) be non-intrusive of the signals function.

The *nanosleep*( ) function shall return a value of 0 on success and −1 on failure or if interrupted.

This latter case is different from *sleep*( ). This was done because the remaining time is returned via an argument structure pointer, *rmtp*, instead of as the return value.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_nanosleep( ), *sleep*( )

XBD **<time.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *nanosleep*( ) function is marked as part of the Timers option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Timers option.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/37 is applied, updating the SEE ALSO section to include the *clock_nanosleep*( ) function.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/63 is applied, correcting text in the RATIONALE section.

Issue 7

SD5-XBD-ERN-33 is applied.

The *nanosleep*( ) function is moved from the Timers option to the Base.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0231 [909] is applied.

NAME

nearbyint, nearbyintf, nearbyintl — floating-point rounding functions

SYNOPSIS

```
#include <math.h>

double nearbyint(double x);
float nearbyintf(float x);
long double nearbyintl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall round their argument to an integer value in floating-point format, using the current rounding direction and without raising the inexact floating-point exception.

RETURN VALUE

MX Upon successful completion, these functions shall return the rounded integer value. The result shall have the same sign as *x*.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, ±0 shall be returned.

If *x* is ±Inf, *x* shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The integral value returned by these functions need not be expressible as an **intmax_t**. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0402 [346,428] is applied.

NAME

newlocale — create or modify a locale object

SYNOPSIS

CX
```
#include <locale.h>

locale_t newlocale(int category_mask, const char *locale,
    locale_t base);
```

DESCRIPTION

The *newlocale*( ) function shall create a new locale object or modify an existing one. If the *base* argument is (**locale_t**)0, a new locale object shall be created, otherwise the locale specified by *base* shall be modified. In the latter case it is unspecified whether the resulting locale object shall be that pointed to by *base* modified in place, or whether that object shall be freed after a new locale object is first created using some values from it.

The *category_mask* argument specifies the locale categories to be set or modified. Values for *category_mask* shall be constructed by a bitwise-inclusive OR of the symbolic constants *LC_CTYPE_MASK*, *LC_NUMERIC_MASK*, *LC_TIME_MASK*, *LC_COLLATE_MASK*, *LC_MONETARY_MASK*, and *LC_MESSAGES_MASK*, or any of the implementation-defined mask values defined in **<locale.h>**.

For each category with the corresponding bit set in *category_mask* the data from the locale named by *locale* shall be used. In the case of modifying an existing locale object, the data from the locale named by *locale* shall replace the existing data within the locale object. If a completely new locale object is created, the data for all sections not requested by *category_mask* shall be taken from the POSIX locale.

The following preset values of *locale* are defined for all settings of *category_mask*:

"POSIX" Specifies the minimal environment for C-language translation called the POSIX locale.

"C" Equivalent to `"POSIX"`.

"" Specifies an implementation-defined native environment. This corresponds to the value of the associated environment variables, *LC_** and *LANG*; see XBD Chapter 7 (on page 127) and Chapter 8 (on page 167).

If the *base* argument is not (**locale_t**)0 and the *newlocale*( ) function call succeeds, the contents of *base* are unspecified. Applications shall ensure that they stop using *base* as a locale object before calling *newlocale*( ). If the function call fails and the *base* argument is not (**locale_t**)0, the contents of *base* shall remain valid and unchanged.

The behavior is undefined if the *base* argument is the special locale object LC_GLOBAL_LOCALE, or is not a valid locale object handle and is not (**locale_t**)0.

RETURN VALUE

Upon successful completion, the *newlocale*( ) function shall return a handle which the caller may use on subsequent calls to *duplocale*( ), *freelocale*( ), and other functions taking a **locale_t** argument.

Upon failure, the *newlocale*( ) function shall return (**locale_t**)0 and set *errno* to indicate the error.

ERRORS

The *newlocale*( ) function shall fail if:

50201 [ENOMEM] There is not enough memory available to create the locale object or load the
50202 locale data.

50203 [EINVAL] The *category_mask* contains a bit that does not correspond to a valid category.

50204 [ENOENT] For any of the categories in *category_mask*, the locale data is not available.

50205 The *newlocale*( ) function may fail if:

50206 [EINVAL] The *locale* argument is not a valid string pointer.

50207 **EXAMPLES**

50208 **Constructing a Locale Object from Different Locales**

50209 The following example shows the construction of a locale where the *LC_CTYPE* category data
50210 comes from a locale *loc1* and the *LC_TIME* category data from a locale *loc2*:

```
50211 #include <locale.h>
50212 ...
50213 locale_t loc, new_loc;
50214 /* Get the "loc1" data. */
50215 loc = newlocale (LC_CTYPE_MASK, "loc1", (locale_t)0);
50216 if (loc == (locale_t) 0)
50217     abort ();
50218 /* Get the "loc2" data. */
50219 new_loc = newlocale (LC_TIME_MASK, "loc2", loc);
50220 if (new_loc != (locale_t) 0)
50221     /* We don t abort if this fails. In this case this
50222        simply used to unchanged locale object. */
50223     loc = new_loc;
50224 ...
```

50225 **Freeing up a Locale Object**

50226 The following example shows a code fragment to free a locale object created by *newlocale*( ):

```
50227 #include <locale.h>
50228 ...
50229 /* Every locale object allocated with newlocale() should be
50230  * freed using freelocale():
50231  */
50232 locale_t loc;
50233 /* Get the locale. */
50234 loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", (locale_t)0);
50235 /* ... Use the locale object ... */
50236 ...
50237 /* Free the locale object resources. */
50238 freelocale (loc);
```


APPLICATION USAGE

Handles for locale objects created by the *newlocale*( ) function should either be released by a corresponding call to *freelocale*( ), or be used as a base locale to another *newlocale*( ) call.

The special locale object LC_GLOBAL_LOCALE must not be passed for the *base* argument, even when returned by the *uselocale*( ) function.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

duplocale( ), *freelocale*( ), *getlocalename_l*( ), *uselocale*( )

XBD Chapter 7 (on page 127), Chapter 8 (on page 167), **<locale.h>**

CHANGE HISTORY

First released in Issue 7.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0403 [227], XSH/TC1-2008/0404 [283], XSH/TC1-2008/0405 [295], and XSH/TC1-2008/0406 [227] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0232 [781] and XSH/TC2-2008/0233 [673] are applied.

Issue 8

Austin Group Defect 1220 is applied, adding *getlocalename_l*( ) to the SEE ALSO section.

Austin Group Defect 1243 is applied, clarifying the handling of a non-zero *base* argument.

Austin Group Defect 1264 is applied, changing ``default locale'' to ``POSIX locale''.

NAME

nextafter, nextafterf, nextafterl, nexttoward, nexttowardf, nexttowardl — next representable floating-point number

SYNOPSIS

```
#include <math.h>

double nextafter(double x, double y);
float nextafterf(float x, float y);
long double nextafterl(long double x, long double y);
double nexttoward(double x, long double y);
float nexttowardf(float x, long double y);
long double nexttowardl(long double x, long double y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *nextafter*( ), *nextafterf*( ), and *nextafterl*( ) functions shall compute the next representable floating-point value following *x* in the direction of *y*. Thus, if *y* is less than *x*, *nextafter*( ) shall return the largest representable floating-point number less than *x*. The *nextafter*( ), *nextafterf*( ), and *nextafterl*( ) functions shall return *y* if *x* equals *y*.

The *nexttoward*( ), *nexttowardf*( ), and *nexttowardl*( ) functions shall be equivalent to the corresponding *nextafter*( ) functions, except that the second parameter shall have type **long double** and the functions shall return *y* converted to the type of the function if *x* equals *y*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the next representable floating-point value following *x* in the direction of *y*.

If *x*==*y*, *y* (of the type *x*) shall be returned.

MX Even though underflow or overflow can occur, the returned value shall be independent of the current rounding direction mode.

If *x* is finite and the correct function value would overflow, a range error shall occur and ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (with the same sign as *x*) shall be returned as appropriate for the return type of the function.

MX If *x* or *y* is NaN, a NaN shall be returned.

MX If *x*!=*y* and the correct function value is subnormal, zero, or underflows, a range error shall occur, and

MXX the correct function value (if representable) or

MX 0.0 shall be returned.

ERRORS

These functions shall fail if:

Range Error — The correct value overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression

(*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

MX Range Error — The correct value is subnormal or underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

When **<tgmath.h>** is included, note that the return type of *nextafter*( ) depends on the generic typing deduced from both arguments, while the return type of *nexttoward*( ) depends only on the generic typing of the first argument.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( )

XBD Section 4.23 (on page 109), **<math.h>**, **<tgmath.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The *nextafter*( ) function is no longer marked as an extension.

The *nextafterf*( ), *nextafterl*( ), *nexttoward*( ), *nexttowardf*( ), and *nexttowardl*( ) functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0407 [68] and XSH/TC1-2008/0408 [357] are applied.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

NAME

nftw — walk a file tree

SYNOPSIS

XSI

```
#include <ftw.h>

int nftw(const char *path, int (*fn)(const char *,
    const struct stat *, int, struct FTW *), int fd_limit, int flags);
```

DESCRIPTION

The *nftw*( ) function shall recursively descend the directory hierarchy rooted in *path*. The argument *flags* is a bitwise-inclusive OR of zero or more of the following flags:

FTW_CHDIR — If set, *nftw*( ) shall change the current working directory to each directory as it reports files in that directory. If clear, *nftw*( ) shall not change the current working directory.

FTW_DEPTH — If set, *nftw*( ) shall report all files in a directory before reporting the directory itself. If clear, *nftw*( ) shall report any directory before reporting the files in that directory.

FTW_MOUNT — If set, *nftw*( ) shall only report files that have the same device ID (*st_dev*) as *path* and shall not descend below directories that have a different device ID than *path*. If clear, *nftw*( ) shall report all files encountered during the walk, unless FTW_XDEV is set.

FTW_PHYS — If set, *nftw*( ) shall perform a physical walk and shall not follow symbolic links.

FTW_XDEV — If set, *nftw*( ) shall not descend below directories that have a different device ID (*st_dev*) than *path*; that is, when a directory with a different device ID is encountered, *nftw*( ) shall report the directory itself (unless FTW_MOUNT is set) but shall not report any files below the directory. If clear, *nftw*( ) shall report all files encountered during the walk, unless FTW_MOUNT is set.

Note: If both FTW_MOUNT and FTW_XDEV are set, *nftw*( ) obeys both flags but the end result is the same as if FTW_XDEV were clear.

If FTW_PHYS is clear and FTW_DEPTH is set, *nftw*( ) shall follow links instead of reporting them, but shall not report any directory that would be a descendant of itself. If FTW_PHYS is clear and FTW_DEPTH is clear, *nftw*( ) shall follow links instead of reporting them, but shall not report the contents of any directory that would be a descendant of itself.

At each file it encounters, *nftw*( ) shall call the user-supplied function *fn* with four arguments:

- The first argument is the pathname of the object.
- The second argument is a pointer to the **stat** buffer containing information on the object, filled in as if *fstatat*( ), *stat*( ), or *lstat*( ) had been called to retrieve the information.
- The third argument is an integer giving additional information. Its value is one of the following:

  FTW_D — The object is a directory.

  FTW_DNR — The object is a directory that cannot be read. The *fn* function shall not be called for any of its descendants.

  FTW_DP — The object is a directory and subdirectories have been visited. (This condition shall only occur if the FTW_DEPTH flag is included in *flags*.)

FTW_F The object is a non-directory file.

FTW_NS The *stat*( ) function failed on the object because of lack of appropriate permission. The **stat** buffer passed to *fn* is undefined. Failure of *stat*( ) for any other reason is considered an error and *nftw*( ) shall return –1.

FTW_SL The object is a symbolic link. (This condition shall only occur if the FTW_PHYS flag is included in *flags*.)

FTW_SLN The object is a symbolic link that does not name an existing file. The **stat** buffer passed to *fn* shall contain information on the symbolic link. (This condition shall only occur if the FTW_PHYS flag is not included in *flags*.)

- The fourth argument is a pointer to an **FTW** structure. The value of **base** is the offset of the object's filename in the pathname passed as the first argument to *fn*. The value of **level** indicates depth relative to the root of the walk, where the root level is 0.

The results are unspecified if the application-supplied *fn* function does not preserve the current working directory.

The argument *fd_limit* sets the maximum number of file descriptors that shall be used by *nftw*( ) while traversing the file tree. At most one file descriptor shall be used for each directory level. The FD_CLOEXEC flag shall be set on any file descriptor opened by *nftw*( ) (see **<fcntl.h>**) not including those opened by the user-supplied *fn* function. Every file descriptor opened by *nftw*( ) not including those opened by the user-supplied *fn* function shall be closed before *nftw*( ) returns.

The *nftw*( ) function need not be thread-safe.

RETURN VALUE

The *nftw*( ) function shall continue until the first of the following conditions occurs:

- An invocation of *fn* shall return a non-zero value, in which case *nftw*( ) shall return that value.
- The *nftw*( ) function detects an error other than [EACCES] (see FTW_DNR and FTW_NS above), in which case *nftw*( ) shall return –1 and set *errno* to indicate the error.
- The tree is exhausted, in which case *nftw*( ) shall return 0.

ERRORS

The *nftw*( ) function shall fail if:

[EACCES] Search permission is denied for any component of *path* or read permission is denied for *path*, or *fn* returns –1 and does not reset *errno*.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG] The length of a component of a pathname is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of *path* names an existing file that is neither a directory nor a symbolic link to a directory.

[EOVERFLOW] A field in the **stat** structure cannot be represented correctly in the current programming environment for one or more files found in the file hierarchy.

The *nftw*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[EMFILE] All file descriptors available to the process are currently open.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

[ENFILE] Too many files are currently open in the system.

In addition, *errno* may be set if the function pointed to by *fn* causes *errno* to be set.

EXAMPLES

The following program traverses the directory tree under the path named in its first command-line argument, or under the current directory if no argument is supplied. It displays various information about each file. The second command-line argument can be used to specify characters that control the value assigned to the *flags* argument when calling *nftw*( ).

```
#include <ftw.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>

static int
display_info(const char *fpath, const struct stat *sb,
    int tflag, struct FTW *ftwbuf)
{
    printf("%-3s %2d %7jd   %-40s %d %s\n",
        (tflag == FTW_D) ?   "d"   : (tflag == FTW_DNR) ? "dnr" :
        (tflag == FTW_DP) ?  "dp"  : (tflag == FTW_F) ?
            (S_ISBLK(sb->st_mode) ? "f b" :
             S_ISCHR(sb->st_mode) ? "f c" :
             S_ISFIFO(sb->st_mode) ? "f p" :
             S_ISREG(sb->st_mode) ? "f r" :
             S_ISSOCK(sb->st_mode) ? "f s" : "f ?") :
        (tflag == FTW_NS) ?  "ns"  : (tflag == FTW_SL) ?  "sl" :
        (tflag == FTW_SLN) ? "sln" : "?", ftwbuf->level,
        (intmax_t) ((tflag == FTW_NS) ? -1 : sb->st_size),
        fpath, ftwbuf->base, fpath + ftwbuf->base);
    return 0;           /* To tell nftw() to continue */
}

int
main(int argc, char *argv[])
{
    int flags = 0;

    if (argc > 2 && strchr(argv[2], 'd') != NULL)
        flags |= FTW_DEPTH;
    if (argc > 2 && strchr(argv[2], 'p') != NULL)
        flags |= FTW_PHYS;

    if (nftw((argc < 2) ? "." : argv[1], display_info, 20, flags) == -1)
    {
```



```
50477                perror("nftw");
50478                exit(EXIT_FAILURE);
50479            }

50480            exit(EXIT_SUCCESS);
50481        }
```

APPLICATION USAGE

50482 50483 50484 50485 50486 50487 50488 The *nftw*( ) function may allocate dynamic storage during its operation. If *nftw*( ) is forcibly terminated, such as by *longjmp*( ) or *siglongjmp*( ) being executed by the function pointed to by *fn* or an interrupt routine, *nftw*( ) does not have a chance to free that storage, so it remains permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have the function pointed to by *fn* return a non-zero value at its next invocation.

50489 50490 50491 50492 50493 50494 50495 50496 When restricting the walk to files on one file system, it can sometimes be desirable for the crossing points themselves to be reported and sometimes for them not to be reported. (Crossing points are mount points and, if FTW_PHYS is clear, symbolic links to directories on other file systems.) With FTW_XDEV *nftw*( ) reports them and with FTW_MOUNT it does not. However, with FTW_MOUNT it also does not report symbolic links to non-directory files on other file systems (if FTW_PHYS is clear). If there is a need for an application to exclude crossing points but include symbolic links to non-directory files on other file systems, this can be achieved by using FTW_XDEV and performing a check such as the following in the function pointed to by *fn*:

```
50497            if (tflag == FTW_D && sb->st_dev != saved_dev)
50498               return 0;
```

50499 (where `saved_dev` is the *st_dev* value for *path*).

RATIONALE

50500 50501 50502 50503 50504 50505 50506 Earlier versions of this standard did not make clear that, as well as not reporting them, FTW_MOUNT prevents descent below directories that have a different device ID than *path* if they are encountered by following a symbolic link (rather than by being a mount point). This meant that if such a directory contained any symbolic links to files with the same device ID as *path*, *nftw*( ) with FTW_PHYS clear was required to report them. However, this was not how *nftw*( ) implementations behaved and the standard has been amended to match existing practice.

FUTURE DIRECTIONS

50507 50508 None.

SEE ALSO

50509 50510 *fdopendir*( ), *fstatat*( ), *readdir*( )

50511 XBD **<fcntl.h>**, **<ftw.h>**

CHANGE HISTORY

50512 50513 First released in Issue 4, Version 2.

Issue 5

50514 50515 Moved from X/OPEN UNIX extension to BASE.

50516 In the DESCRIPTION, the definition of the *depth* argument is clarified.

Issue 6

50517 50518 The Open Group Base Resolution bwg97-003 is applied.

50519 The ERRORS section is updated as follows:

- The wording of the mandatory [ELOOP] error condition is updated.
- A second optional [ELOOP] error condition is added.
- The [EOVERFLOW] mandatory error condition is added.

Text is added to the DESCRIPTION to say that the *nftw*( ) function need not be reentrant and that the results are unspecified if the application-supplied *fn* function does not preserve the current working directory.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/64 is applied, changing the argument *depth* to *fd_limit* throughout and changing ``to a maximum of 5 levels deep'' to ``using a maximum of 5 file descriptors'' in the EXAMPLES section.

Issue 7

Austin Group Interpretations 1003.1-2001 #143 and #156 are applied.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XBD-ERN-61 is applied.

APPLICATION USAGE is added and the EXAMPLES section is replaced with a new example.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0409 [403], XSH/TC1-2008/0410 [324], and XSH/TC1-2008/0411 [403] are applied.

Issue 8

Austin Group Defect 368 is applied, adding a requirement for the FD_CLOEXEC flag to be set.

Austin Group Defect 1121 is applied, changing the description of FTW_SLN and the handling of FTW_NS in the EXAMPLES section.

Austin Group Defect 1133 is applied, adding FTW_XDEV.

Austin Group Defect 1210 is applied, changing the description of FTW_MOUNT and the RATIONALE section.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

NAME

ngettext, ngettext_l — message handling functions

SYNOPSIS

```
#include <libintl.h>

char *ngettext(const char *msgid, const char *msgid_plural,
    unsigned long int n);
char *ngettext_l(const char *msgid, const char *msgid_plural,
    unsigned long int n, locale_t locale);
```

DESCRIPTION

Refer to *gettext*.

NAME

nice — change the nice value of a process

SYNOPSIS

XSI
```
#include <unistd.h>

int nice(int incr);
```

DESCRIPTION

The *nice*( ) function shall add the value of *incr* to the nice value of the calling process. A nice value of a process is a non-negative number for which a more positive value shall result in less favorable scheduling.

A maximum nice value of 2*{NZERO}–1 and a minimum nice value of 0 shall be imposed by the system. Requests for values above or below these limits shall result in the nice value being set to the corresponding limit. Only a process with appropriate privileges can lower the nice value.

PS | TPS Calling the *nice*( ) function has no effect on the priority of processes or threads with policy SCHED_FIFO or SCHED_RR. The effect on processes or threads with other scheduling policies is implementation-defined.

The nice value set with *nice*( ) shall be applied to the process. If the process is multi-threaded, the nice value shall affect all system scope threads in the process.

As –1 is a permissible return value in a successful situation, an application wishing to check for error situations should set *errno* to 0, then call *nice*( ), and if it returns –1, check to see whether *errno* is non-zero.

RETURN VALUE

Upon successful completion, *nice*( ) shall return the new nice value –{NZERO}. Otherwise, –1 shall be returned, the nice value of the process shall not be changed, and *errno* shall be set to indicate the error.

ERRORS

The *nice*( ) function shall fail if:

[EPERM] The *incr* argument is negative and the calling process does not have appropriate privileges.

EXAMPLES

Changing the Nice Value

The following example adds the value of the *incr* argument, –20, to the nice value of the calling process.

```
#include <unistd.h>
...
int incr = -20;
int ret;

ret = nice(incr);
```

APPLICATION USAGE

None.

50593 **RATIONALE**

50594 None.

50595 **FUTURE DIRECTIONS**

50596 None.

50597 **SEE ALSO**

50598 *exec*, *getpriority( )*

50599 XBD **<limits.h>**, **<unistd.h>**

50600 **CHANGE HISTORY**

50601 First released in Issue 1. Derived from Issue 1 of the SVID.

50602 **Issue 5**

50603 A statement is added to the description indicating the effects of this function on the different
50604 scheduling policies and multi-threaded processes.

NAME

nl_langinfo, nl_langinfo_l — language information

SYNOPSIS

```
#include <langinfo.h>

char *nl_langinfo(nl_item item);
char *nl_langinfo_l(nl_item item, locale_t locale);
```

DESCRIPTION

The *nl_langinfo*( ) and *nl_langinfo_l*( ) functions shall return a pointer to a string containing information relevant to the particular language or cultural area defined in the current locale, or in the locale represented by *locale*, respectively (see **<langinfo.h>**). The manifest constant names and values of *item* are defined in **<langinfo.h>**. For example:

```
nl_langinfo(ABDAY_1)
```

would return a pointer to the string `"Dom"` if the identified language was Portuguese, and `"Sun"` if the identified language was English.

```
nl_langinfo_l(ABDAY_1, loc)
```

would return a pointer to the string `"Dom"` if the identified language of the locale represented by *loc* was Portuguese, and `"Sun"` if the identified language of the locale represented by *loc* was English.

The *nl_langinfo*( ) function need not be thread-safe.

The behavior is undefined if the *locale* argument to *nl_langinfo_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

In a locale where *langinfo* data is not defined, these functions shall return a pointer to the corresponding string in the POSIX locale. In all locales, these functions shall return a pointer to an empty string if *item* contains an invalid setting.

The application shall not modify the string returned. The pointer returned by *nl_langinfo*( ) might be invalidated or the string content might be overwritten by a subsequent call to *nl_langinfo*( ) in any thread or to *nl_langinfo_l*( ) in the same thread or the initial thread, by subsequent calls to *setlocale*( ) with a category corresponding to the category of *item* (see **<langinfo.h>**) or the category LC_ALL, or by subsequent calls to *uselocale*( ) which change the category corresponding to the category of *item*. The pointer returned by *nl_langinfo_l*( ) might be invalidated or the string content might be overwritten by a subsequent call to *nl_langinfo_l*( ) in the same thread or to *nl_langinfo*( ) in any thread, or by subsequent calls to *freelocale*( ) or *newlocale*( ) which free or modify the locale object that was passed to *nl_langinfo_l*( ). The returned pointer and the string content might also be invalidated if the calling thread is terminated.

ERRORS

No errors are defined.

EXAMPLES

Getting Date and Time Formatting Information

The following example returns a pointer to a string containing date and time formatting information, as defined in the *LC_TIME* category of the current locale.

```
#include <time.h>
#include <langinfo.h>
...
strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
...
```

APPLICATION USAGE

The array pointed to by the return value should not be modified by the program, but may be modified by further calls to these functions.

RATIONALE

The possible interactions between internal data used by *nl_langinfo*( ) and *nl_langinfo_l*( ) are complicated by the fact that *nl_langinfo_l*( ) must be thread-safe but *nl_langinfo*( ) need not be. The various implementation choices are:

1. *nl_langinfo_l*( ) and *nl_langinfo*( ) use separate buffers, or at least one of them does not use an internal string buffer. In this case there are no interactions.
2. *nl_langinfo_l*( ) and *nl_langinfo*( ) share an internal per-thread buffer. There can be interactions, but only in the same thread.
3. *nl_langinfo_l*( ) uses an internal per-thread buffer, and *nl_langinfo*( ) uses (in all threads) the same buffer that *nl_langinfo_l*( ) uses in the initial thread. There can be interactions, but only when *nl_langinfo_l*( ) is called in the initial thread.

FUTURE DIRECTIONS

None.

SEE ALSO

setlocale( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<langinfo.h>**, **<locale.h>**, **<nl_types.h>**

CHANGE HISTORY

First released in Issue 2.

Issue 5

The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

The *nl_langinfo*( ) function is moved from the XSI option to the Base.

The *nl_langinfo_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0412 [302], XSH/TC1-2008/0413 [75], XSH/TC1-2008/0414 [283], XSH/TC1-2008/0415 [75,402], XSH/TC1-2008/0416 [283], and XSH/TC1-2008/0417 [402] are applied.

50684 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0234 [656] is applied.

NAME

nrand48 — generate uniformly distributed pseudo-random non-negative long integers

SYNOPSIS

XSI

```
#include <stdlib.h>

long nrand48(unsigned short xsubi[3]);
```

DESCRIPTION

Refer to *drand48( )*.

NAME

ntohl, ntohs — convert values between host and network byte order

SYNOPSIS

```
#include <arpa/inet.h>

uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

DESCRIPTION

Refer to *htonl( )*.

NAME

open, openat — open file

SYNOPSIS

OH
```
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *path, int oflag, ...);
int openat(int fd, const char *path, int oflag, ...);
```

DESCRIPTION

The *open*( ) function shall establish the connection between a file and a file descriptor. It shall create an open file description that refers to a file and a file descriptor that refers to that open file description. The file descriptor is used by other I/O functions to refer to that file. The *path* argument points to a pathname naming the file.

The *open*( ) function shall return a file descriptor for the named file, allocated as described in Section 2.6 (on page 525). The open file description is new, and therefore the file descriptor shall not share it with any other process in the system. The FD_CLOEXEC file descriptor flag associated with the new file descriptor shall be cleared unless the O_CLOEXEC flag is set in *oflag*. The FD_CLOFORK file descriptor flag associated with the new file descriptor shall be cleared unless the O_CLOFORK flag is set in *oflag*.

The file offset used to mark the current position within the file shall be set to the beginning of the file.

The file status flags and file access modes of the open file description shall be set according to the value of *oflag*.

Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in **<fcntl.h>**. Applications shall specify exactly one of the first five values (file access modes) below in the value of *oflag*:

O_EXEC — Open for execute only (non-directory files). If *path* names a directory and O_EXEC is not the same value as O_SEARCH, *open*( ) shall fail.

O_RDONLY — Open for reading only.

O_RDWR — Open for reading and writing. If *path* names a FIFO, and the implementation does not support opening a FIFO for simultaneous read and write, then *open*( ) shall fail.

O_SEARCH — Open directory for search only. If *path* names a non-directory file and O_SEARCH is not the same value as O_EXEC, *open*( ) shall fail.

O_WRONLY — Open for writing only.

Any combination of the following may be used:

O_APPEND — If set, the file offset shall be set to the end of the file prior to each write.

O_CLOEXEC — If set, the FD_CLOEXEC flag for the new file descriptor shall be set.

O_CLOFORK — If set, the FD_CLOFORK flag for the new file descriptor shall be set.

O_CREAT — If the file exists, this flag has no effect except as noted under O_EXCL below. Otherwise, if O_DIRECTORY is not set the file shall be created as a regular file; the user ID of the file shall be set to the effective user ID of the process; the group ID of the file shall be set to the group ID of the file's parent directory or to the effective group ID of the process; and the access permission bits (see **<sys/stat.h>**) of the file mode shall be set to the value

of the argument following the *oflag* argument taken as type **mode_t** modified as follows: a bitwise AND is performed on the file-mode bits and the corresponding bits in the complement of the process' file mode creation mask. Thus, all bits in the file mode whose corresponding bit in the file mode creation mask is set are cleared. When bits other than the file permission bits are set, the effect is unspecified. The argument following the *oflag* argument does not affect whether the file is open for reading, writing, or for both. Implementations shall provide a way to initialize the file's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the file's group ID to the effective group ID of the calling process.

O_DIRECTORY — If *path* resolves to a non-directory file, fail and set *errno* to [ENOTDIR].

SIO O_DSYNC — Write I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion.

O_EXCL — If O_CREAT and O_EXCL are set, *open*( ) shall fail if the file exists. The check for the existence of the file and the creation of the file if it does not exist shall be atomic with respect to other threads executing *open*( ) naming the same filename in the same directory with O_EXCL and O_CREAT set. If O_EXCL and O_CREAT are set, and *path* names a symbolic link, *open*( ) shall fail and set *errno* to [EEXIST], regardless of the contents of the symbolic link. If O_EXCL is set and O_CREAT is not set, the result is undefined.

O_NOCTTY — If set and *path* identifies a terminal device, *open*( ) shall not cause the terminal device to become the controlling terminal for the process. If *path* does not identify a terminal device, O_NOCTTY shall be ignored.

O_NOFOLLOW — If *path* names a symbolic link, fail and set *errno* to [ELOOP].

O_NONBLOCK — When opening a FIFO with O_RDONLY or O_WRONLY set:

- If O_NONBLOCK is set, an *open*( ) for reading-only shall return without delay. An *open*( ) for writing-only shall return an error if no process currently has the file open for reading.
- If O_NONBLOCK is clear, an *open*( ) for reading-only shall block the calling thread until a thread opens the file for writing. An *open*( ) for writing-only shall block the calling thread until a thread opens the file for reading.

When opening a block special or character special file that supports non-blocking opens:

- If O_NONBLOCK is set, the *open*( ) function shall return without blocking for the device to be ready or available. Subsequent behavior of the device is device-specific.
- If O_NONBLOCK is clear, the *open*( ) function shall block the calling thread until the device is ready or available before returning.

Otherwise, the O_NONBLOCK flag shall not cause an error, but it is unspecified whether the file status flags will include the O_NONBLOCK flag.

SIO O_RSYNC Read I/O operations on the file descriptor shall complete at the same level of integrity as specified by the O_DSYNC and O_SYNC flags. If both O_DSYNC and O_RSYNC are set in *oflag*, all I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion. If both O_SYNC and O_RSYNC are set in flags, all I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.

XSI|SIO O_SYNC Write I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.

XSI The O_SYNC flag shall be supported for regular files, even if the Synchronized Input and Output option is not supported.

O_TRUNC If the file exists and is a regular file, and the file is successfully opened O_RDWR or O_WRONLY, its length shall be truncated to 0, and the mode and owner shall be unchanged. It shall have no effect on FIFO special files or terminal device files. Its effect on other file types is implementation-defined. The result of using O_TRUNC without either O_RDWR or O_WRONLY is undefined.

O_TTY_INIT If *path* identifies a terminal device other than a pseudo-terminal, the device is not already open in any process, and either O_TTY_INIT is set in *oflag* or O_TTY_INIT has the value zero, *open*( ) shall set any non-standard **termios** structure terminal parameters to a state that provides conforming behavior (see XBD Section 11.2, on page 205) and initialize the **winsize** structure associated with the terminal to appropriate default settings. It is unspecified whether O_TTY_INIT has any effect if the device is already open in any process. If *path* identifies the subsidiary side of a pseudo-terminal that is not already open in any process, *open*( ) shall set any non-standard **termios** structure terminal parameters to a state that provides conforming behavior and initialize the **winsize** structure associated with the terminal to appropriate default settings, regardless of whether O_TTY_INIT is set. If *path* does not identify a terminal device, O_TTY_INIT shall be ignored.

If O_CREAT and O_DIRECTORY are set and the requested access mode is neither O_WRONLY nor O_RDWR, the result is unspecified.

If O_CREAT is set and the file did not previously exist, upon successful completion, *open*( ) shall mark for update the last data access, last data modification, and last file status change timestamps of the file and the last data modification and last file status change timestamps of the parent directory.

If O_TRUNC is set and the file did previously exist, upon successful completion, *open*( ) shall mark for update the last data modification and last file status change timestamps of the file.

SIO If both the O_SYNC and O_DSYNC flags are set, the effect is as if only the O_SYNC flag was set.

The application shall ensure that it specifies the O_TTY_INIT flag on the first open of a terminal device since system boot or since the device was closed by the process that last had it open. The application need not specify the O_TTY_INIT flag when opening pseudo-terminals. If *path* names the manager side of a pseudo-terminal device, then it is unspecified whether *open*( ) locks the subsidiary side so that it cannot be opened. Conforming applications shall call *unlockpt*( ) before opening the subsidiary side.

The largest value that can be represented correctly in an object of type **off_t** shall be established as the offset maximum in the open file description.

The *openat*( ) function shall be equivalent to the *open*( ) function except in the case where *path* specifies a relative path. In this case the file to be opened is determined relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

The *oflag* parameter and the optional fourth parameter correspond exactly to the parameters of *open*( ).

If *openat*( ) is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used and the behavior shall be identical to a call to *open*( ).

RETURN VALUE

Upon successful completion, these functions shall open the file and return a non-negative integer representing the file descriptor. Otherwise, these functions shall return –1 and set *errno* to indicate the error. If –1 is returned, no files shall be created or modified.

ERRORS

These functions shall fail if:

[EACCES] Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by *oflag* are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created, or O_TRUNC is specified and write permission is denied.

[EEXIST] O_CREAT and O_EXCL are set, and the named file exists.

[EILSEQ] O_CREAT was specified, the file did not exist, and the last pathname component of *path* is not a portable filename and cannot be created in the target directory.

[EINTR] A signal was caught during *open*( ).

[EINVAL] The *path* argument names a FIFO, O_RDWR was specified, and the SIO implementation considers this an error; or synchronized I/O flags were specified and the implementation does not support synchronized I/O for the file.

[EISDIR] The named file is a directory and *oflag* includes O_WRONLY or O_RDWR, or includes O_CREAT without O_DIRECTORY, or includes O_EXEC when O_EXEC is not the same value as O_SEARCH.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument, or O_NOFOLLOW was specified and the *path* argument names a symbolic link.

[EMFILE] All file descriptors available to the process are currently open.

[ENAMETOOLONG] The length of a component of a pathname is longer than {NAME_MAX}.

[ENFILE] The maximum allowable number of files is currently open in the system.

[ENOENT] O_CREAT is not set and a component of *path* does not name an existing file, or O_CREAT is set and a component of the path prefix of *path* does not name an existing file, or *path* points to an empty string.

[ENOENT] or [ENOTDIR]
O_CREAT is set, and the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters. If *path* without the trailing <slash> characters would name an existing file, an [ENOENT] error shall not occur.

[ENOSPC] The directory or file system that would contain the new file cannot be expanded, the file does not exist, and O_CREAT is specified.

[ENOTDIR] A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory; or O_CREAT and O_EXCL are not specified, the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters, and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory; or O_DIRECTORY was specified and the *path* argument names a non-directory file; or the *path* argument names a non-directory file and O_SEARCH was specified when O_SEARCH is not the same value as O_EXEC.

[ENXIO] O_NONBLOCK is set, the named file is a FIFO, O_WRONLY is set, and no process has the file open for reading.

[ENXIO] The named file is a character special or block special file, and the device associated with this special file does not exist.

[EOVERFLOW] The named file is a regular file and the size of the file cannot be represented correctly in an object of type **off_t**.

[EROFS] The named file resides on a read-only file system and either O_WRONLY, O_RDWR, O_CREAT (if the file does not exist), or O_TRUNC is set in the *oflag* argument.

The *openat*( ) function shall fail if:

[EACCES] The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

[EBADF] The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

[ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

These functions may fail if:

XSI [EAGAIN] The *path* argument names the subsidiary side of a pseudo-terminal device that is locked.

[EINVAL] The value of the *oflag* argument is not valid.

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

[EOPNOTSUPP] The *path* argument names a socket.

[ETXTBSY] The file is a pure procedure (shared text) file that is being executed and *oflag* is O_WRONLY or O_RDWR.

EXAMPLES

Opening a File for Writing by the Owner

The following example opens the file **/tmp/file**, either by creating it (if it does not already exist), or by truncating its length to 0 (if it does exist). In the former case, if the call creates a new file, the access permission bits in the file mode of the file are set to permit reading and writing by the owner, and to permit reading only by group members and others.

If the call to *open*( ) is successful, the file is opened for writing.

```
#include <fcntl.h>
...
int fd;
mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
char *pathname = "/tmp/file";
...
fd = open(pathname, O_WRONLY | O_CREAT | O_TRUNC, mode);
...
```

Opening a File Using an Existence Check

The following example uses the *open*( ) function to try to create the **LOCKFILE** file and open it for writing. Since the *open*( ) function specifies the O_EXCL flag, the call fails if the file already exists. In that case, the program assumes that someone else is updating the password file and exits.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

#define LOCKFILE "/etc/ptmp"
...
int pfd; /* Integer for file descriptor returned by open() call. */
...
if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
{
    fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
    exit(1);
}
...
```


Opening a File for Writing

The following example opens a file for writing, creating the file if it does not already exist. If the file does exist, the system truncates the file to zero bytes.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

#define LOCKFILE "/etc/ptmp"
...
int pfd;
char pathname[PATH_MAX+1];
...
if ((pfd = open(pathname, O_WRONLY | O_CREAT | O_TRUNC,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
{
    perror("Cannot open output file\n"); exit(1);
}
...
```

APPLICATION USAGE

POSIX.1-2024 does not require that terminal parameters be automatically set to any state on first open, nor that they be reset after the last close. It is possible for a non-conforming application to leave a terminal device in a state where the next process to use that device finds it in a non-conforming state, but has no way of determining this. To ensure that the device is set to a conforming initial state, applications which perform a first open of a terminal (other than a pseudo-terminal) should do so using the O_TTY_INIT flag to set the parameters associated with the terminal to a conforming state.

Except as specified in this volume of POSIX.1-2024, the flags allowed in *oflag* are not mutually-exclusive and any number of them may be used simultaneously. Not all combinations of flags make sense. For example, using O_SEARCH | O_CREAT will successfully open a pre-existing directory for searching, but if there is no existing file by that name, then it is unspecified whether a regular file will be created. Likewise, if a non-directory file descriptor is successfully returned, it is unspecified whether that descriptor will have execute permissions as if by O_EXEC (note that it is unspecified whether O_EXEC and O_SEARCH have the same value).

The O_CLOEXEC and O_CLOFORK flags of *open*( ) are necessary to avoid a data race in multi-threaded applications. Without O_CLOFORK, a file descriptor is leaked into a child process created by one thread in the window between another thread creating a file descriptor with *open*( ) and then using *fcntl*( ) to set the FD_CLOFORK flag. Without O_CLOEXEC, a file descriptor intentionally inherited by child processes is similarly leaked into an executed program if FD_CLOEXEC is not set atomically.

RATIONALE

Some implementations permit opening FIFOs with O_RDWR. Since FIFOs could be implemented in other ways, and since two file descriptors can be used to the same effect, an implementation is allowed to reject the use of O_RDWR on a FIFO.

See *getgroups*( ) about the group of a newly created file.

The use of *open*( ) to create a regular file is preferable to the use of *creat*( ), because the latter is redundant and included only for historical reasons.

The use of the O_TRUNC flag on FIFOs and directories (pipes cannot be *open*( )-ed) must be permissible without unexpected side-effects (for example, *creat*( ) on a FIFO must not remove

data). Since terminal special files might have type-ahead data stored in the buffer, O_TRUNC should not affect their content, particularly if a program that normally opens a regular file should open the current controlling terminal instead. Other file types, particularly implementation-defined ones, are left implementation-defined.

POSIX.1-2024 permits [EACCES] to be returned for conditions other than those explicitly listed.

The O_NOCTTY flag was added to allow applications to avoid unintentionally acquiring a controlling terminal as a side-effect of opening a terminal file. This volume of POSIX.1-2024 does not specify how a controlling terminal is acquired, but it allows an implementation to provide this on *open*( ) if the O_NOCTTY flag is not set and other conditions specified in XBD Chapter 11 (on page 199) are met.

In historical implementations the value of O_RDONLY is zero. Because of that, it is not possible to detect the presence of O_RDONLY and another option. Future implementations should encode O_RDONLY and O_WRONLY as bit flags so that:

```
O_RDONLY | O_WRONLY == O_RDWR
```

O_EXEC and O_SEARCH are specified as two of the five file access modes. Since O_EXEC does not apply to directories, and O_SEARCH only applies to directories, their values need not be distinct. Although this standard requires *open*( ) to fail on an attempt to use O_EXEC on a directory, or O_SEARCH on a non-directory, this only applies in implementations where the two modes have distinct values. Since O_RDONLY has historically had the value zero, implementations are not able to distinguish between O_SEARCH and O_SEARCH | O_RDONLY, and similarly for O_EXEC.

In general, the *open*( ) function follows the symbolic link if *path* names a symbolic link. However, the *open*( ) function, when called with O_CREAT and O_EXCL, is required to fail with [EEXIST] if *path* names an existing symbolic link, even if the symbolic link refers to a nonexistent file. This behavior is required so that privileged applications can create a new file in a known location without the possibility that a symbolic link might cause the file to be created in a different location.

For example, a privileged application that must create a file with a predictable name in a user-writable directory, such as the user's home directory, could be compromised if the user creates a symbolic link with that name that refers to a nonexistent file in a system directory. If the user can influence the contents of a file, the user could compromise the system by creating a new system configuration or spool file that would then be interpreted by the system. The test for a symbolic link which refers to a nonexisting file must be atomic with the creation of a new file.

In addition, the *open*( ) function refuses to open non-directories if the O_DIRECTORY flag is set. This avoids race conditions whereby a user might compromise the system by substituting a hard link to a sensitive file (e.g., a device or a FIFO) while a privileged application is running, where opening a file even for read access might have undesirable side-effects.

In addition, the *open*( ) function does not follow symbolic links if the O_NOFOLLOW flag is set. This avoids race conditions whereby a user might compromise the system by substituting a symbolic link to a sensitive file (e.g., a device) while a privileged application is running, where opening a file even for read access might have undesirable side-effects.

The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required that implementations provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also supporting a way to set the group ID to the effective group ID of the creating process. Conforming applications should not assume which group ID will be used. If it matters, an application can use *chown*( ) to set the

group ID after the file is created, or determine under what conditions the implementation will set the desired group ID.

The purpose of the *openat*( ) function is to enable opening files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *open*( ), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *openat*( ) function it can be guaranteed that the opened file is located relative to the desired directory. Some implementations use the *openat*( ) function for other purposes as well. In some cases, if the *oflag* parameter has the O_XATTR bit set, the returned file descriptor provides access to extended attributes. This functionality is not standardized here.

Implementations are encouraged to have *open*( ) and *openat*( ) report an [EILSEQ] error if *oflag* includes O_CREAT, the file did not previously exist, and the last component of *path* contains any bytes that have the encoded value of a <newline> character.

FUTURE DIRECTIONS

A future version of this standard may add an O_NOCLOBBER flag, specified as follows, for use by shells when the *noclobber* option is set (see XRAT Section C.2.7.2, on page 3891):

O_NOCLOBBER — If O_CREAT and O_NOCLOBBER are set, *open*( ) shall fail if the file exists and is either a regular file or a symbolic link that resolves to a regular file. The check for the existence and type of the file and the creation of the file if it does not exist shall be atomic with respect to other threads executing *open*( ) naming the same filename in the same directory with O_NOCLOBBER and O_CREAT set or with O_EXCL and O_CREAT set. If O_NOCLOBBER and O_CREAT are set, and the file exists and is either a non-regular file or a symbolic link that resolves to a non-regular file, the file shall be opened as if neither flag was set. If O_NOCLOBBER and O_CREAT are set, and *path* names a symbolic link that does not resolve to an existing file, an empty file shall be created such that *path* resolves to the newly created file. If O_NOCLOBBER is set and O_CREAT is not set, the result is undefined.

SEE ALSO

chmod( ), *close*( ), *creat*( ), *dirfd*( ), *dup*( ), *exec*, *fcntl*( ), *fdopendir*( ), *link*( ), *lseek*( ), *mkdtemp*( ), *mknod*( ), *read*( ), *symlink*( ), *umask*( ), *unlockpt*( ), *write*( )

XBD Chapter 11 (on page 199), **<fcntl.h>**, **<sys/stat.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Large File Summit extensions are added.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

- In the DESCRIPTION, O_CREAT is amended to state that the group ID of the file is set to the group ID of the file's parent directory or to the effective group ID of the process. This is a FIPS requirement.
- In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files.
- In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support large files.
- The [ENXIO] mandatory error condition is added.
- The [EINVAL], [ENAMETOOLONG], and [ETXTBSY] optional error conditions are added.

The DESCRIPTION and ERRORS sections are updated so that items related to the optional XSI STREAMS Option Group are marked.

The following changes were made to align with the IEEE P1003.1a draft standard:

- An explanation is added of the effect of the O_CREAT and O_EXCL flags when the path refers to a symbolic link.
- The [ELOOP] optional error condition is added.

The normative text is updated to avoid use of the term ``must'' for application requirements.

The DESCRIPTION of O_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.

Issue 7

Austin Group Interpretations 1003.1-2001 #113 and #143 are applied.

Austin Group Interpretation 1003.1-2001 #144 is applied, adding the O_TTY_INIT flag.

Austin Group Interpretation 1003.1-2001 #171 is applied, adding support to set the FD_CLOEXEC flag atomically at *open*( ), and adding the F_DUPFD_CLOEXEC flag.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

This page is revised and the *openat*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Functionality relating to the XSI STREAMS option is marked obsolescent.

Changes are made related to support for finegrained timestamps.

Changes are made to allow a directory to be opened for searching.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0418 [292], XSH/TC1-2008/0419 [141], XSH/TC1-2008/0420 [461], XSH/TC1-2008/0421 [390], XSH/TC1-2008/0422 [146], XSH/TC1-2008/0423 [324], XSH/TC1-2008/0424 [292], XSH/TC1-2008/0425 [278], XSH/TC1-2008/0426 [278], XSH/TC1-2008/0427 [291], and XSH/TC1-2008/0428 [307] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0235 [873], XSH/TC2-2008/0236 [835], XSH/TC2-2008/0237 [847], XSH/TC2-2008/0238 [817], XSH/TC2-2008/0239 [835], XSH/TC2-2008/0240 [847], XSH/TC2-2008/0241 [822], XSH/TC2-2008/0242 [817], and XSH/TC2-2008/0243 [943] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 293 is applied, adding the [EILSEQ] error.

Austin Group Defects 658 and 1665 are applied, restricting the allowed behaviors when O_EXEC is used on a directory, or O_SEARCH on a non-directory file, or O_RDWR on a FIFO, so that the requirements for O_EXCL still apply.

Austin Group Defect 1016 is applied, changing the FUTURE DIRECTIONS section.

Austin Group Defect 1151 is applied, changing the description of O_TTY_INIT to include requirements relating to the **winsize** structure.

Austin Group Defect 1318 is applied, adding FD_CLOFORK and O_CLOFORK.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1466 is applied, changing the terminology used for pseudo-terminal devices.

NAME

open_memstream, open_wmemstream — open a dynamic memory buffer stream

SYNOPSIS

CX
```
#include <stdio.h>

FILE *open_memstream(char **bufp, size_t *sizep);

#include <wchar.h>

FILE *open_wmemstream(wchar_t **bufp, size_t *sizep);
```

DESCRIPTION

The *open_memstream*( ) and *open_wmemstream*( ) functions shall create an I/O stream associated with a dynamically allocated memory buffer. The stream shall be opened for writing and shall be seekable.

The stream associated with a call to *open_memstream*( ) shall be byte-oriented.

The stream associated with a call to *open_wmemstream*( ) shall be wide-oriented.

The stream shall maintain a current position in the allocated buffer and a current buffer length. The position shall be initially set to zero (the start of the buffer). Each write to the stream shall start at the current position and move this position by the number of successfully written bytes for *open_memstream*( ) or the number of successfully written wide characters for *open_wmemstream*( ). The length shall be initially set to zero. If a write moves the position to a value larger than the current length, the current length shall be set to this position. In this case a null character for *open_memstream*( ) or a null wide character for *open_wmemstream*( ) shall be appended to the current buffer. For both functions the terminating null is not included in the calculation of the buffer length.

After a successful *fflush*( ) or *fclose*( ), the pointer referenced by *bufp* shall contain the address of the buffer, and the variable pointed to by *sizep* shall contain the smaller of the current buffer length and the number of bytes for *open_memstream*( ), or the number of wide characters for *open_wmemstream*( ), between the beginning of the buffer and the current file position indicator.

The *fseek*( ) and *fseeko*( ) functions can be used to set the file position beyond the current buffer length. It is implementation-defined whether this extends the buffer to the new length. If it extends the buffer, the added buffer contents shall be set to null bytes for *open_memstream*( ), or null wide characters for *open_wmemstream*( ); if it does not extend the buffer, then if data is later written at this point, the buffer contents in the gap shall be set to null bytes for *open_memstream*( ), or null wide characters for *open_wmemstream*( ). If *fseek*( ) or *fseeko*( ) is called with SEEK_END as the *whence* argument, it is implementation-defined whether the file position is adjusted relative to the current buffer length or relative to the buffer size that would be set by an *fflush*( ) call made immediately before the *fseek*( ) or *fseeko*( ) call.

After a successful *fflush*( ) the pointer referenced by *bufp* and the variable referenced by *sizep* remain valid only until the next write operation on the stream or a call to *fclose*( ).

After a successful *fclose*( ), the pointer referenced by *bufp* can be passed to *free*( ).

RETURN VALUE

Upon successful completion, these functions shall return a pointer to the object controlling the stream. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

ERRORS

These functions shall fail if:

[EMFILE] {STREAM_MAX} streams are currently open in the calling process.

These functions may fail if:

[EINVAL] *bufp* or *sizep* are NULL.

[EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

[ENOMEM] Memory for the stream or the buffer could not be allocated.

EXAMPLES

```
#include <stdio.h>
#include <stdlib.h>

int
main (void)
{
    FILE *stream;
    char *buf;
    size_t len;
    off_t eob;

    stream = open_memstream (&buf, &len);
    if (stream == NULL)
        /* handle error */ ;
    fprintf (stream, "hello my world");
    fflush (stream);
    printf ("buf=%s, len=%zu\n", buf, len);
    eob = ftello(stream);
    fseeko (stream, 0, SEEK_SET);
    fprintf (stream, "good-bye");
    fseeko (stream, eob, SEEK_SET);
    fclose (stream);
    printf ("buf=%s, len=%zu\n", buf, len);
    free (buf);
    return 0;
}
```

This program produces the following output:

```
buf=hello my world, len=14
buf=good-bye world, len=14
```

APPLICATION USAGE

The buffer created by these functions should be freed by the application after closing the stream, by means of a call to *free*( ).

RATIONALE

These functions are similar to *fmemopen*( ) except that the memory is always allocated dynamically by the function, and the stream is opened only for output.

FUTURE DIRECTIONS

None.

SEE ALSO

fclose( ), *fdopen*( ), *fflush*( ), *fmemopen*( ), *fopen*( ), *free*( ), *freopen*( )

XBD **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 7.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0244 [588] and XSH/TC2-2008/0245 [586] are applied.

Issue 8

Austin Group Defect 1406 is applied, clarifying the behavior of *fseek*( ) and *fseeko*( ) on streams created by *open_memstream*( ) and *open_wmemstream*( ).

NAME

openat — open file relative to directory file descriptor

SYNOPSIS

```
#include <fcntl.h>

int openat(int fd, const char *path, int oflag, ...);
```

DESCRIPTION

Refer to *open( )*.

NAME

opendir — open directory associated with file descriptor

SYNOPSIS

```
#include <dirent.h>

DIR *opendir(const char *dirname);
```

DESCRIPTION

Refer to *fdopendir( )*.

NAME

openlog — open a connection to the logging facility

SYNOPSIS

XSI

```
#include <syslog.h>

void openlog(const char *ident, int logopt, int facility);
```

DESCRIPTION

Refer to *closelog( )*.

NAME

optarg, opterr, optind, optopt — options parsing variables

SYNOPSIS

```
#include <unistd.h>

extern char *optarg;
extern int opterr, optind, optopt;
```

DESCRIPTION

Refer to *getopt*( ).

NAME

pathconf — get configurable pathname variables

SYNOPSIS

```
#include <unistd.h>

long pathconf(const char *path, int name);
```

DESCRIPTION

Refer to *fpathconf( )*.

NAME

pause — suspend the thread until a signal is received

SYNOPSIS

```
#include <unistd.h>

int pause(void);
```

DESCRIPTION

The *pause*( ) function shall suspend the calling thread until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process.

If the action is to terminate the process, *pause*( ) shall not return.

If the action is to execute a signal-catching function, *pause*( ) shall return after the signal-catching function returns.

RETURN VALUE

Since *pause*( ) suspends thread execution indefinitely unless interrupted by a signal, there is no successful completion return value. A value of –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *pause*( ) function shall fail if:

[EINTR] A signal is caught by the calling process and control is returned from the signal-catching function.

EXAMPLES

None.

APPLICATION USAGE

Many common uses of *pause*( ) have timing windows. The scenario involves checking a condition related to a signal and, if the signal has not occurred, calling *pause*( ). When the signal occurs between the check and the call to *pause*( ), the process often blocks indefinitely. The *sigprocmask*( ) and *sigsuspend*( ) functions can be used to avoid this type of problem.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sigsuspend( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The APPLICATION USAGE section is added.

NAME

pclose — close a pipe stream to or from a process

SYNOPSIS

CX
```
#include <stdio.h>

int pclose(FILE *stream);
```

DESCRIPTION

The *pclose*( ) function shall close a stream that was opened by *popen*( ), wait for the command to terminate, and return the termination status of the process that was running the command language interpreter. However, if a call caused the termination status to be unavailable to *pclose*( ), then *pclose*( ) shall return –1 with *errno* set to [ECHILD] to report this situation. This can happen if the application calls one of the following functions:

- *wait*( )
- *waitpid*( ) with a *pid* argument less than or equal to 0 or equal to the process ID of the command line interpreter
- Any other function not defined in this volume of POSIX.1-2024 that could do one of the above

In any case, *pclose*( ) shall not return before the child process created by *popen*( ) has terminated.

If the command language interpreter cannot be executed, the child termination status returned by *pclose*( ) shall be as if the command language interpreter terminated using *exit*(127) or *_exit*(127).

The *pclose*( ) function shall not affect the termination status of any child of the calling process other than the one created by *popen*( ) for the associated stream.

If the argument *stream* to *pclose*( ) is not a pointer to a stream created by *popen*( ), the result of *pclose*( ) is undefined.

If a thread is canceled during execution of *pclose*( ), the behavior is undefined.

RETURN VALUE

Upon successful return, *pclose*( ) shall return the termination status of the command language interpreter. Otherwise, *pclose*( ) shall return –1 and set *errno* to indicate the error.

ERRORS

The *pclose*( ) function shall fail if:

[ECHILD] The status of the child process could not be obtained, as described above.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

There is a requirement that *pclose*( ) not return before the child process terminates. This is intended to disallow implementations that return [EINTR] if a signal is received while waiting. If *pclose*( ) returned before the child terminated, there would be no way for the application to discover which child used to be associated with the stream, and it could not do the cleanup itself.

If the stream pointed to by *stream* was not created by *popen*( ), historical implementations of

pclose( ) return –1 without setting *errno*. To avoid requiring *pclose*( ) to set *errno* in this case, POSIX.1-2024 makes the behavior unspecified. An application should not use *pclose*( ) to close any stream that was not created by *popen*( ).

Some historical implementations of *pclose*( ) either block or ignore the signals SIGINT, SIGQUIT, and SIGHUP while waiting for the child process to terminate. Since this behavior is not described for the *pclose*( ) function in POSIX.1-2024, such implementations are not conforming. Also, some historical implementations return [EINTR] if a signal is received, even though the child process has not terminated. Such implementations are also considered non-conforming.

Consider, for example, an application that uses:

```
popen("command", "r")
```

to start *command*, which is part of the same application. The parent writes a prompt to its standard output (presumably the terminal) and then reads from the *popen*( )ed stream. The child reads the response from the user, does some transformation on the response (pathname expansion, perhaps) and writes the result to its standard output. The parent process reads the result from the pipe, does something with it, and prints another prompt. The cycle repeats. Assuming that both processes do appropriate buffer flushing, this would be expected to work.

To conform to POSIX.1-2024, *pclose*( ) must use *waitpid*( ), or some similar function, instead of *wait*( ).

See the RATIONALE for *popen*( ) for a sample implementation of *pclose*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

fork( ), *popen*( ), *wait*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0246 [632] is applied.

Issue 8

Austin Group Defect 411 is applied, changing the RATIONALE section.

NAME

perror — write error messages to standard error

SYNOPSIS

```
#include <stdio.h>

void perror(const char *s);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *perror*( ) function shall map the error number accessed through the symbol *errno* to a language-dependent error message, which shall be written to the standard error stream as follows:

- First (if *s* is not a null pointer and the character pointed to by *s* is not the null byte), the string pointed to by *s* followed by a <colon> and a <space>.
- Then an error message string followed by a <newline>.

The contents of the error message strings shall be the same as those returned by *strerror*( ) with argument *errno*.

CX The *perror*( ) function shall mark for update the last data modification and last file status change timestamps of the file associated with the standard error stream at some time between its successful completion and *exit*( ), *abort*( ), or the completion of *fflush*( ) or *fclose*( ) on *stderr*.

The *perror*( ) function shall not change the orientation of the standard error stream.

On error, *perror*( ) shall set the error indicator for the stream to which *stderr* points, and shall set *errno* to indicate the error.

Since no value is returned, an application wishing to check for error situations should call *clearerr*(*stderr*) before calling *perror*( ), then if *ferror*(*stderr*) returns non-zero, the value of *errno* indicates which error occurred.

RETURN VALUE

The *perror*( ) function shall not return a value.

ERRORS

CX Refer to *fputc*( ).

EXAMPLES

Printing an Error Message for a Function

The following example replaces *bufptr* with a buffer that is the necessary size. If an error occurs, the *perror*( ) function prints a message and the program exits.

```
#include <stdio.h>
#include <stdlib.h>
...
char *bufptr;
size_t szbuf;
...
if ((bufptr = malloc(szbuf)) == NULL) {
    perror("malloc"); exit(2);
}
```


. . .

APPLICATION USAGE

Application writers may prefer to use alternative interfaces instead of *perror*( ), such as *strerror_r*( ) in combination with *fprintf*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fprintf( ), *fputc*( ), *psiginfo*( ), *strerror*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

A paragraph is added to the DESCRIPTION indicating that *perror*( ) does not change the orientation of the standard error stream.

Issue 6

Extensions beyond the ISO C standard are marked.

Issue 7

SD5-XSH-ERN-95 is applied.

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0429 [389,401], XSH/TC1-2008/0430 [389], and XSH/TC1-2008/0431 [389,401] are applied.

NAME

pipe, pipe2 — create an interprocess channel

SYNOPSIS

```
#include <unistd.h>

int pipe(int fildes[2]);
int pipe2(int fildes[2], int flag);
```

DESCRIPTION

The *pipe*( ) function shall create a pipe and place two file descriptors, one each into the arguments *fildes*[0] and *fildes*[1], that refer to the open file descriptions for the read and write ends of the pipe, respectively. The file descriptors shall be allocated as described in Section 2.6 (on page 525). The FD_CLOEXEC and FD_CLOFORK flags shall be clear on both file descriptors. The O_NONBLOCK flag shall be clear on both open file descriptions. (The *fcntl*( ) function can be used to set this flag.)

Data can be written to the file descriptor *fildes*[1] and read from the file descriptor *fildes*[0]. A read on the file descriptor *fildes*[0] shall access data written to the file descriptor *fildes*[1] on a first-in-first-out basis. It is unspecified whether *fildes*[0] is also open for writing and whether *fildes*[1] is also open for reading.

A process has the pipe open for reading (correspondingly writing) if it has a file descriptor open that refers to the read end, *fildes*[0] (write end, *fildes*[1]).

The pipe's user ID shall be set to the effective user ID of the calling process.

The pipe's group ID shall be set to the effective group ID of the calling process.

Upon successful completion, *pipe*( ) shall mark for update the last data access, last data modification, and last file status change timestamps of the pipe.

The *pipe2*( ) function shall be equivalent to the *pipe*( ) function, except that the state of O_NONBLOCK on the new file descriptions and FD_CLOEXEC and FD_CLOFORK on the new file descriptors shall be determined solely by the *flag* argument, which can be constructed from a bitwise-inclusive OR of flags from the following list (provided by **<fcntl.h>**):

O_CLOEXEC Atomically set the FD_CLOEXEC flag on both new file descriptors.

O_CLOFORK Atomically set the FD_CLOFORK flag on both new file descriptors.

O_NONBLOCK Set the O_NONBLOCK file status flag on both new file descriptions.

RETURN VALUE

Upon successful completion, 0 shall be returned; otherwise, –1 shall be returned and *errno* set to indicate the error, no file descriptors shall be allocated and the contents of *fildes* shall be left unmodified.

ERRORS

The *pipe*( ) and *pipe2*( ) functions shall fail if:

[EMFILE] All, or all but one, of the file descriptors available to the process are currently open.

[ENFILE] The number of simultaneously open files in the system would exceed a system-imposed limit.

The *pipe2*( ) function may fail if:

[EINVAL] The value of the *flag* argument is invalid.

EXAMPLES

Using a Pipe to Pass Data Between a Parent Process and a Child Process

The following example demonstrates the use of a pipe to transfer data between a parent process and a child process. Error handling is excluded, but otherwise this code demonstrates good practice when using pipes: after the *fork*( ) the two processes close the unused ends of the pipe before they commence transferring data.

```
#include <stdlib.h>
#include <unistd.h>
...

int fildes[2];
const int BSIZE = 100;
char buf[BSIZE];
ssize_t nbytes;
int status;

status = pipe(fildes);
if (status == -1 ) {
    /* an error occurred */
    ...
}

switch (fork()) {
case -1: /* Handle error */
    break;

case 0:  /* Child - reads from pipe */
    close(fildes[1]);                       /* Write end is unused */
    nbytes = read(fildes[0], buf, BSIZE);   /* Get data from pipe */
    /* At this point, a further read would see end-of-file ... */
    close(fildes[0]);                       /* Finished with pipe */
    exit(EXIT_SUCCESS);

default:  /* Parent - writes to pipe */
    close(fildes[0]);                       /* Read end is unused */
    write(fildes[1], "Hello world\n", 12);  /* Write data on pipe */
    close(fildes[1]);                       /* Child will see EOF */
    exit(EXIT_SUCCESS);
}
```

APPLICATION USAGE

None.

RATIONALE

The wording carefully avoids using the verb ``to open'' in order to avoid any implication of use of *open*( ); see also *write*( ).

The O_CLOEXEC and O_CLOFORK flags of *pipe2*( ) are necessary to avoid a data race in multi-threaded applications. Without O_CLOFORK, a file descriptor is leaked into a child process created by one thread in the window between another thread creating a file descriptor with *pipe*( ) and then using *fcntl*( ) to set the FD_CLOFORK flag. Without O_CLOEXEC, a file descriptor intentionally inherited by child processes is similarly leaked into an executed

program if FD_CLOEXEC is not set atomically.

Since pipes are often used for communication between a parent and child process, O_CLOFORK has to be used with care in order for the pipe to be usable. If the parent will be writing and the child will be reading, O_CLOFORK should be used when creating the pipe, and then *fcntl*( ) should be used to clear FD_CLOFORK for the read side of the pipe. This prevents the write side from leaking into other children, ensuring the child will get end-of-file when the parent closes the write side (although the read side can still be leaked). If the parent will be reading and the child will be writing, there is no way to prevent the write side being leaked (short of preventing other threads from creating child processes) in order to ensure the parent gets end-of-file when the child closes the write side, and so the two processes should use an alternative method of indicating the end of communications.

Arranging for FD_CLOEXEC to be set appropriately is more straightforward. The parent should use O_CLOEXEC when creating the pipe and the child should clear FD_CLOEXEC on the side to be passed to the new program before calling an *exec* family function to execute it.

The O_NONBLOCK flag is for convenience in avoiding additional *fcntl*( ) calls.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.6 (on page 525), *fcntl*( ), *read*( ), *write*( )

XBD **<fcntl.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to indicate that certain dispositions of *fildes*[0] and *fildes*[1] are unspecified.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/65 is applied, adding the example to the EXAMPLES section.

Issue 7

SD5-XSH-ERN-156 is applied, updating the DESCRIPTION to state the setting of the pipe's user ID and group ID.

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0247 [835] and XSH/TC2-2008/0248 [467,835] are applied.

Issue 8

Austin Group Defects 411, 1318, and 1577 are applied, adding *pipe2*( ) and FD_CLOFORK.

Austin Group Defect 1576 is applied, adding the word ``respectively''.

NAME

poll, ppoll — input/output multiplexing

SYNOPSIS

```
#include <poll.h>

int poll(struct pollfd fds[], nfds_t nfds, int timeout);
int ppoll(struct pollfd fds[], nfds_t nfds,
    const struct timespec *restrict timeout,
    const sigset_t *restrict sigmask);
```

DESCRIPTION

The *ppoll*( ) function provides applications with a mechanism for multiplexing input/output over a set of file descriptors. For each member of the array pointed to by *fds*, *ppoll*( ) shall examine the given file descriptor for the event(s) specified in *events*. The number of **pollfd** structures in the *fds* array is specified by *nfds*. The *ppoll*( ) function shall identify those file descriptors on which an application can make an attempt to read or write data without blocking, or on which certain events have occurred.

The *poll*( ) function shall be equivalent to the *ppoll*( ) function, except as follows:

- For the *poll*( ) function, the timeout period is given in milliseconds in an argument of type **int**, whereas for the *ppoll*( ) function the timeout period is given in seconds and nanoseconds via an argument of type pointer to **struct timespec**. A *timeout* of –1 for *poll*( ) shall be equivalent to passing a null pointer for the *timeout* for *ppoll*( ).
- The *poll*( ) function has no *sigmask* argument; it shall behave as *ppoll*( ) does when *sigmask* is a null pointer.

The *fds* argument specifies the file descriptors to be examined and the events of interest for each file descriptor. It is a pointer to an array with one member for each open file descriptor of interest. The array's members are **pollfd** structures within which *fd* specifies an open file descriptor and *events* and *revents* are bitmasks constructed by OR'ing a combination of the following event flags:

POLLIN	The file descriptor is ready for reading data other than high-priority data.
POLLRDNORM	The file descriptor is ready for reading normal data.
POLLRDBAND	The file descriptor is ready for reading priority data.
POLLPRI	The file descriptor is ready for reading high-priority data.
POLLOUT	The file descriptor is ready for writing normal data.
POLLWRNORM	Equivalent to POLLOUT.
POLLWRBAND	The file descriptor is ready for writing priority data.
POLLERR	An error condition is present on the file descriptor. All error conditions that arise solely from the state of the object underlying the open file description and would be diagnosed by a return of –1 from a *read*( ) or *write*( ) call on the file descriptor shall be reported as a POLLERR event. This flag is only valid in the *revents* bitmask; it shall be ignored in the *events* member.
POLLHUP	A device has been disconnected, or a pipe or FIFO has been closed by the last process that had it open for writing. Once set, the hangup state of a FIFO shall persist until some process opens the FIFO for writing or until all read-only file descriptors for the FIFO are closed. This event and POLLOUT are mutually-exclusive. However, this event and POLLIN, POLLRDNORM,

POLLRDBAND, or POLLPRI are not mutually-exclusive. This flag is only valid in the *revents* bitmask; it shall be ignored in the *events* member.

POLLNVAL The specified *fd* value is not an open file descriptor. This flag is only valid in the *revents* member; it shall be ignored in the *events* member.

A file descriptor shall be considered ready for reading when a call to an input function with O_NONBLOCK clear would not block, whether or not the function would transfer data successfully. (The function might return data, an end-of-file indication, or an error other than one indicating that it is blocked, and in each of these cases the descriptor is considered ready for reading.) A file descriptor shall be considered ready for writing when a call to an output function with O_NONBLOCK clear would not block, whether or not the function would transfer data successfully. How much data could be written without blocking depends upon the object underlying the open file description and its current state.

The significance and semantics of normal, priority, and high-priority data are file and device-specific. The semantics of device disconnection are device-specific.

If the value of *fd* is less than 0, *events* shall be ignored, and *revents* shall be set to 0 in that entry on return from *poll*( ) or *ppoll*( ).

In each **pollfd** structure, *poll*( ) or *ppoll*( ) shall clear the *revents* member, except that where the application requested a report on a condition by setting one of the bits of *events* listed above, *poll*( ) or *ppoll*( ) shall set the corresponding bit in *revents* if the requested condition is true. In addition, *poll*( ) or *ppoll*( ) shall set the POLLHUP, POLLERR, and POLLNVAL flag in *revents* if the condition is true, even if the application did not set the corresponding bit in *events*.

The *timeout* argument controls how long the *poll*( ) or *ppoll*( ) function shall wait before timing out. If the *timeout* argument is positive for *poll*( ) or not a null pointer for *ppoll*( ), it specifies a maximum interval to wait for the poll to complete. If the specified time interval expires without any of the defined events having occurred, the function shall return. If the *timeout* argument is –1 for *poll*( ) or a null pointer for *ppoll*( ), then the call shall block indefinitely until at least one descriptor meets the specified criteria or until the call is interrupted. To effect a poll, the application shall ensure that the *timeout* argument for *poll*( ) is 0, or for *ppoll*( ) is not a null pointer and points to a zero-valued **timespec** structure.

Implementations may place limitations on the maximum timeout interval supported. All implementations shall support a maximum timeout interval of at least 31 days for *ppoll*( ). If the *timeout* argument specifies a timeout interval greater than the implementation-defined maximum value, the maximum value shall be used as the actual timeout value. Implementations may also place limitations on the granularity of timeout intervals. If the requested timeout interval requires a finer granularity than the implementation supports, the actual timeout interval shall be rounded up to the next supported value.

The *poll*( ) and *ppoll*( ) functions shall not be affected by the O_NONBLOCK flag.

The *poll*( ) and *ppoll*( ) functions shall support regular files, terminal and pseudo-terminal devices, FIFOs, pipes, and sockets. The behavior of *poll*( ) and *ppoll*( ) on elements of *fds* that refer to other types of file is unspecified.

Regular files shall always poll TRUE for reading and writing.

A file descriptor for a socket that is listening for connections shall indicate that it is ready for reading, once connections are available. A file descriptor for a socket that is connecting asynchronously shall indicate that it is ready for writing, once a connection has been established.

Provided the application does not perform any action that results in unspecified or undefined behavior, the value of the *fd* and *events* members of each element of *fds* shall not be modified by

poll( ) or *ppoll*( ).

If *sigmask* is not a null pointer, the *ppoll*( ) function shall replace the signal mask of the caller by the set of signals pointed to by *sigmask* before examining the descriptors, and shall restore the signal mask of the calling thread before returning. If a signal is unmasked as a result of the signal mask being altered by *ppoll*( ), and a signal-catching function is called for that signal during the execution of the *ppoll*( ) function, and SA_RESTART is clear for the interrupting signal, then

- If none of the defined events have occurred on any selected file descriptor, *ppoll*( ) shall immediately fail with the [EINTR] error after the signal-catching function returns.
- If one or more of the defined events have occurred, it is unspecified whether *ppoll*( ) behaves the same as if none of the events had occurred (failing with [EINTR] as above) or behaves the same as if it was not interrupted (returning the total number of **pollfd** structures that have selected events).

If a thread is canceled during a *ppoll*( ) call, it is unspecified whether the signal mask in effect when executing the registered cleanup functions is the original signal mask or the signal mask installed as part of the *ppoll*( ) call.

RETURN VALUE

Upon successful completion, a non-negative value shall be returned. A positive value shall indicate the total number of **pollfd** structures that have selected events (that is, those for which the *revents* member is non-zero). A value of 0 shall indicate that the call timed out and no file descriptors have been selected. Upon failure, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *poll*( ) and *ppoll*( ) functions shall fail if:

[EAGAIN] The allocation of internal data structures failed but a subsequent request may succeed.

[EINTR] A signal was caught during *poll*( ) or *ppoll*( ).

[EINVAL] The *nfds* argument is greater than {OPEN_MAX}.

The *ppoll*( ) function shall fail if:

[EINVAL] An invalid timeout interval was specified.

EXAMPLES

None.

APPLICATION USAGE

Other than the difference in the precision of the requested timeout, the following *ppoll*( ) call:

```
ready = ppoll(&fds, nfds, tmo_p, &sigmask);
```

is equivalent to atomically executing the following calls:

```
sigset_t origmask;
int timeout;

timeout = (tmo_p == NULL) ? -1 :
    (tmo_p->tv_sec * 1000 + tmo_p->tv_nsec / 1000000);
pthread_sigmask(SIG_SETMASK, &sigmask, &origmask);
ready = poll(&fds, nfds, timeout);
pthread_sigmask(SIG_SETMASK, &origmask, NULL);
```


When a *poll*( ) or *ppoll*( ) call indicates a file descriptor is ready for reading, this means that if an attempt to read data had been made at the time that the status of the file descriptor was checked, it would have returned at least one byte of data, an end-of-file indication, or an error, without blocking (even if O_NONBLOCK is clear). When a *poll*( ) or *ppoll*( ) call indicates that a file descriptor is ready for writing, this means that if an attempt to write one byte of data had been made at the time that the status of the file descriptor was checked, it would have written that byte or returned an error, without blocking. However, if an attempt to write more than one byte had been made, it might have blocked (if O_NONBLOCK is clear). In both cases, by the time the call returns and a subsequent I/O operation is attempted, the state of the file descriptor might have changed (for example, because another thread read or wrote some data) and, if O_NONBLOCK is clear, there is no guarantee that the operation will not block (unless it would not block for some other reason, such as setting MIN=0 and TIME=0 for a terminal in non-canonical mode). Therefore it is recommended that applications always set O_NONBLOCK on file descriptors whose readiness for I/O they query with *poll*( ) or *ppoll*( ).

The error conditions specified for *read*( ) and *write*( ) that are reported as POLLERR events are only those that arise solely from the state of the object underlying the open file description. They do not include, for example, [EAGAIN] as this relates to the state of the open file description not (solely) the object underlying it.

Application writers should note that repeating a *poll*( ) or *ppoll*( ) call which indicated that I/O was possible on one or more of the file descriptors given, without causing some change to the state, either by altering the *fds* array or causing appropriate input or output to occur on at least one file descriptor indicated as ready, will result in ``busy waiting''—a subsequent call will always return immediately indicating the same (or perhaps more) events as the previous one.

RATIONALE

The POLLHUP event does not occur for FIFOs just because the FIFO is not open for writing. It only occurs when the FIFO is closed by the last writer and persists until some process opens the FIFO for writing or until all read-only file descriptors for the FIFO are closed.

Code which wants to avoid the ambiguity of the signal mask for thread cancellation handlers can install an additional cancellation handler which resets the signal mask to the expected value:

```
void cleanup(void *arg)
{
    sigset_t *ss = (sigset_t *) arg;
    pthread_sigmask(SIG_SETMASK, ss, NULL);
}
int call_ppoll(struct pollfd fds[], nfds_t nfds,
    const struct timespec *restrict timeout,
    const sigset_t *restrict sigmask)
{
    sigset_t oldmask;
    int result;
    pthread_sigmask(SIG_SETMASK, NULL, &oldmask);
    pthread_cleanup_push(cleanup, &oldmask);
    result = ppoll(fds, nfds, timeout, sigmask);
    pthread_cleanup_pop(0);
    return result;
}
```


FUTURE DIRECTIONS

None.

SEE ALSO

pselect( ), *read*( ), *write*( )

XBD **<poll.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The description of POLLWRBAND is updated.

Issue 6

Text referring to sockets is added to the DESCRIPTION.

Functionality relating to the XSI STREAMS Option Group is marked.

The Open Group Corrigendum U055/3 is applied, updating the DESCRIPTION of POLLWRBAND.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/66 is applied, correcting the spacing in the EXAMPLES section.

Issue 7

Austin Group Interpretation 1003.1-2001 #209 is applied, clarifying the POLLHUP event.

The *poll*( ) function is moved from the XSI option to the Base.

Functionality relating to the XSI STREAMS option is marked obsolescent.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0249 [623] and XSH/TC2-2008/0250 [683] are applied.

Issue 8

Austin Group Defect 1263 is applied, adding *ppoll*( ).

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1448 is applied, aligning the wording relating to file descriptor readiness with *pselect*( ) and changing the APPLICATION USAGE section.

NAME

popen — initiate pipe streams to or from a process

SYNOPSIS

CX
```
#include <stdio.h>

FILE *popen(const char *command, const char *mode);
```

DESCRIPTION

The *popen*( ) function shall execute the command specified by the string *command*. It shall create a pipe between the calling program and the executed command, and shall return a pointer to a stream that can be used to either read from or write to the pipe.

The environment of the executed command shall be as if a child process were created within the *popen*( ) call using the *fork*( ) function, and the child invoked the *sh* utility using the call:

```
execl(<shell path>, "sh", "-c", "--", command, (char *)0);
```

where **<***shell path***>** is an unspecified pathname for the *sh* utility. It is implementation-defined whether the handlers registered with *pthread_atfork*( ) are called as part of the creation of the child process.

The *popen*( ) function shall ensure that any streams from previous *popen*( ) calls that remain open in the parent process are closed in the new child process, regardless of the FD_CLOEXEC or FD_CLOFORK status of the file descriptor underlying those streams.

The *mode* argument to *popen*( ) is a string that specifies I/O mode:

1. If *mode* starts with `'r'`, when the child process is started, its file descriptor STDOUT_FILENO shall be the writable end of the pipe, and the file descriptor *fileno*(*stream*) in the calling process, where *stream* is the stream pointer returned by *popen*( ), shall be the readable end of the pipe. The FD_CLOFORK flag shall be cleared on both the STDOUT_FILENO file descriptor passed to the child process and the file descriptor underlying the returned stream.
2. If *mode* starts with `'w'`, when the child process is started its file descriptor STDIN_FILENO shall be the readable end of the pipe, and the file descriptor *fileno*(*stream*) in the calling process, where *stream* is the stream pointer returned by *popen*( ), shall be the writable end of the pipe. The FD_CLOFORK flag shall be cleared on both the STDOUT_FILENO file descriptor passed to the child process and the file descriptor underlying the returned stream.
3. If *mode* includes a second character of `'e'`, then the file descriptor underlying the stream returned to the calling process by *popen*( ) shall have the FD_CLOEXEC flag atomically set. Additionally, if the implementation creates the file descriptor for use by the child process from within the parent process, then that file descriptor shall have the FD_CLOEXEC flag atomically set within the parent process. If *mode* does not have a second character, the FD_CLOEXEC flag of the underlying file descriptor returned by *popen*( ) shall be clear.
4. If *mode* is any other value, the result is unspecified.

After *popen*( ), both the parent and the child process shall be capable of executing independently before either terminates.

Pipe streams are byte-oriented.

RETURN VALUE

Upon successful completion, *popen*( ) shall return a pointer to an open stream that can be used to read or write to the pipe. Otherwise, it shall return a null pointer and may set *errno* to indicate the error.

ERRORS

The *popen*( ) function shall fail if:

[EMFILE] {STREAM_MAX} streams are currently open in the calling process.

The *popen*( ) function may fail if:

[EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

[EINVAL] The *mode* argument is invalid.

The *popen*( ) function may also set *errno* values as described by *fork*( ) or *pipe*( ).

EXAMPLES

Using popen( ) to Obtain a List of Files from the ls Utility

The following example demonstrates the use of *popen*( ) and *pclose*( ) to execute the command *ls** in order to obtain a list of files in the current directory:

```
#include <stdio.h>
...

FILE *fp;
int status;
char path[PATH_MAX];

fp = popen("ls *", "r");
if (fp == NULL)
    /* Handle error */;

while (fgets(path, PATH_MAX, fp) != NULL)
    printf("%s", path);

status = pclose(fp);
if (status == -1) {
    /* Error reported by pclose() */
    ...
} else {
    /* Use macros described under wait() to inspect `status' in order
       to determine success/failure of command executed by popen() */
    ...
}
```

APPLICATION USAGE

Since open files are shared, a mode `'r'` command can be used as an input filter and a mode `'w'` command as an output filter.

Buffered reading before opening an input filter may leave the standard input of that filter mispositioned. Similar problems with an output filter may be prevented by careful buffer flushing; for example, with *fflush*( ).

A stream opened by *popen*( ) should be closed by *pclose*( ).

The behavior of *popen*( ) is specified for values of *mode* of `"r"`, `"w"`, `"re"`, and `"we"`. Other

modes such as "rb" and "wb" might be supported by specific implementations, but these would not be portable features. Note that historical implementations of *popen*( ) only check to see if the first character of *mode* is 'r'. Thus, a *mode* of "robert the robot" would be treated as *mode* "r", and a *mode* of "anything else" would be treated as *mode* "w".

If the application calls *waitpid*( ) or *waitid*( ) with a *pid* argument greater than 0, and it still has a stream that was called with *popen*( ) open, it must ensure that *pid* does not refer to the process started by *popen*( ).

To determine whether or not the environment specified in the Shell and Utilities volume of POSIX.1-2024 is present, use the function call:

```
sysconf(_SC_2_VERSION)
```

(See *sysconf*( )).

RATIONALE

The *popen*( ) function should not be used by programs that have set user (or group) ID privileges. The *fork*( ) and *exec* family of functions (except *execlp*( ) and *execvp*( )), should be used instead. This prevents any unforeseen manipulation of the environment of the user that could cause execution of commands not anticipated by the calling program.

If the original and *popen*( )ed processes both intend to read or write or read and write a common file, and either will be using FILE-type C functions (*fread*( ), *fwrite*( ), and so on), the rules for sharing file handles must be observed (see Section 2.5.1, on page 522).

The 'e' mode modifier to *popen*( ) is necessary to avoid a data race in multi-threaded applications. Without it, the parent's file descriptor is leaked into a second child process created by one thread in the window between another thread creating the pipe via *popen*( ) then using *fileno*( ) and *fcntl*( ) on the result. Also, if the *popen*( ) implementation temporarily has the child's file descriptor open within the parent, then that file descriptor could also be leaked if it is not atomically FD_CLOEXEC for the duration in which it is open in the parent.

The standard only requires that the implementation atomically set FD_CLOEXEC on file descriptors created in the parent process when the 'e' mode modifier is in effect; implementations may also do so when the 'e' modifier is not in use, provided that the FD_CLOEXEC bit is eventually cleared before *popen*( ) completes, however, this is not required because any application worried about the potential file descriptor leak will already be using the 'e' modifier.

Implementations are encouraged to add support for a "wf" mode which creates the pipe as if by calling *pipe2*( ) with the O_CLOFORK flag and then clearing FD_CLOFORK for the read side of the pipe. This prevents the write side from leaking into child processes created by other threads, ensuring the child created by *popen*( ) will get end-of-file when the parent closes the write side (although the read side can still be leaked). Unfortunately there is no way (short of temporarily preventing other threads from creating child processes, or implementing an atomic create-pipe-and-fork system call) to implement an "rf" mode with the equivalent guarantee that the child created by *popen*( ) will be the only writer. Therefore multi-threaded applications that do not have complete control over process creation cannot rely on getting end-of-file on the stream and need to use an alternative method of indicating the end of communications.

Although the standard is clear that a conforming application should not call *popen*( ) when file descriptor 0 or 1 is closed, implementations are encouraged to handle these cases correctly.

The following two examples demonstrate possible implementations of *popen*( ) using other standard functions. These examples are designed to show FD_CLOEXEC handling rather than all aspects of thread safety, and implementations are encouraged to improve the locking

mechanism around the state list to be more efficient, as well as to be more robust if file descriptor 0 or 1 is returned as either part of the pipe. Also, remember that other implementations are possible, including one that uses an implementation-specific means of creating a pipe between parent and child where the parent process never has access to the child's end of the pipe. Both of these examples make use of the following helper functions, documented but not implemented here, to do the bookkeeping necessary to properly close all file descriptors created by other *popen*( ) calls regardless of their FD_CLOEXEC or FD_CLOFORK status:

```
/* Obtain mutual exclusion lock, so that no concurrent popen() or
   pclose() calls are simultaneously modifying the list of tracked
   children. */
static void popen_lock(void);

/* Release mutual exclusion lock, without changing errno. */
static void popen_unlock(void);

/* Add the pid and stream pair to the list of tracked children, prior
   to any code that can clear FD_CLOEXEC on the file descriptor
   associated with stream. To be used while holding the lock. */
static void popen_add_pair(FILE *stream, pid_t pid);

/* Given a stream, return the associated pid, or -1 with errno set if
   the stream was not created by popen(). To be used while holding
   the lock. */
static pid_t popen_get_pid(FILE *stream);

/* Remove stream and its corresponding pid from the list of tracked
   children. To be used while holding the lock. */
static void popen_remove(FILE *stream);

/* If stream is NULL, return the first tracked child; otherwise,
   return the next tracked child. Return NULL if all tracked children
   have been returned. To be used while holding the lock. */
static FILE *popen_next(FILE *stream);
```

The first example is based on *fork*( ):

```
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <unistd.h>
FILE *popen(const char *command, const char *mode)
{
  int fds[2];
  pid_t pid;
  FILE *stream;
  int target = mode[0] == 'w'; /* index of fds used by parent */

  /* Validate mode */
  if ((mode[0] != 'w' && mode[0] != 'r') ||
      mode[1 + (mode[1] == 'e')]) {
    errno = EINVAL;
    return NULL;
  }

  /* Create pipe and stream with FD_CLOEXEC set */
```



```
51966           if (pipe2(fds, O_CLOEXEC) < 0)
51967             return NULL;
51968           stream = fdopen(fds[target], mode);
51969           if (!stream) {
51970             int saved = errno;
51971             close(fds[0]);
51972             close(fds[1]);
51973             errno = saved;
51974             return NULL;
51975           }

51976           /* Create child process */
51977           popen_lock();
51978           pid = fork();
51979           if (pid < 0) {
51980             int saved = errno;
51981             close(fds[!target]);
51982             fclose(stream);
51983             popen_unlock();
51984             errno = saved;
51985             return NULL;
51986           }

51987           /* Child process. */
51988           if (!pid) {
51989             FILE *tracked = popen_next(NULL);
51990             while (tracked) {
51991               int fd = fileno(tracked);
51992               if (fd < 0 || close(fd))
51993                 _exit(127);
51994               tracked = popen_next(tracked);
51995             }
51996             target = mode[0] == 'r'; /* Opposite fd in the child */
51997             /* Use dup2 or fcntl to clear FD_CLOEXEC on child's descriptor,
51998                FD_CLOEXEC will take care of the rest of fds[]. */
51999             if (fds[target] != target) {
52000               if (dup2(fds[target], target) != target)
52001                 _exit(127);
52002             } else {
52003               int flags = fcntl(fds[target], F_GETFD);
52004               if (flags < 0 ||
52005                   fcntl(fds[target], F_SETFD, flags & ~FD_CLOEXEC) < 0)
52006                 _exit(127);
52007             }
52008             execl("/bin/sh", "sh", "-c", "--", command, NULL);
52009             _exit(127);
52010           }

52011           /* Parent process. From here on out, the close and fcntl system
52012              calls are assumed to pass, since all inputs are valid and do not
52013              require allocating any fds or memory. Besides, excluding
52014              failures due to undefined behavior (such as another thread
52015              closing an fd it knows nothing about), cleanup from any defined
```



```
52016             failures would require stopping and reaping the child process,
52017             which may have worse consequences. */
52018           close(fds[!target]);  popen_add_pair(stream, pid);
52019           popen_unlock();
52020           if (mode[1] != 'e') {
52021             int flags = fcntl(fds[target], F_GETFD);
52022             if (flags >= 0)
52023               fcntl(fds[target], F_SETFD, flags & ~FD_CLOEXEC);
52024           }
52025           return stream;
52026         }
```

52027 The second example is based on *posix_spawn*( ):

```
52028         #include <stdio.h>
52029         #include <errno.h>
52030         #include <fcntl.h>
52031         #include <sys/wait.h>
52032         #include <unistd.h>
52033         #include <spawn.h>
52034         extern char **environ;
52035         FILE *popen(const char *command, const char *mode)
52036         {
52037           int fds[2];
52038           pid_t pid;
52039           FILE *stream;
52040           int target = mode[0] == 'w'; /* index of fds used by parent */
52041           const char *argv[] = { "sh", "-c", "--", command, NULL };
52042           posix_spawn_file_actions_t actions;
52043           int saved;
52044           FILE *tracked;

52045           /* Validate mode */
52046           if ((mode[0] != 'w' && mode[0] != 'r') ||
52047               mode[1 + (mode[1] == 'e')]) {
52048             errno = EINVAL;
52049             return NULL;
52050           }

52051           /* Create pipe and stream with FD_CLOEXEC set */
52052           if (pipe2(fds, O_CLOEXEC) < 0)
52053             return NULL;
52054           stream = fdopen(fds[target], mode);
52055           if (!stream) {
52056             saved = errno;
52057             close(fds[0]);
52058             close(fds[1]);
52059             errno = saved;
52060             return NULL;
52061           }

52062           /* Create child process */
52063           if (posix_spawn_file_actions_init(&actions)) {
52064             saved = errno;
```



```
52065             goto spawnerr1;
52066           }
52067           popen_lock();
52068           tracked = popen_next(NULL);
52069           while (tracked) {
52070             int fd = fileno(tracked);
52071             if (fd < 0 || posix_spawn_file_actions_addclose(&actions, fd))
52072               goto spawnerr2;
52073             tracked = popen_next(tracked);
52074           }
52075           if (posix_spawn_file_actions_adddup2(&actions, fds[!target], !target))
52076             goto spawnerr2;
52077           if (posix_spawn(&pid, "/bin/sh", &actions, NULL, (char **)argv,
52078         environ)) {
52079         spawnerr2:
52080             saved = errno;
52081             posix_spawn_file_actions_destroy(&actions);
52082             popen_unlock();
52083         spawnerr1:
52084             close(fds[!target]);
52085             fclose(stream);
52086             errno = saved;
52087             return NULL;
52088           }
52089           /* From here on out, system calls are assumed to pass, since all
52090              inputs are valid and do not require allocating any fds or memory.
52091              Besides, excluding failures due to undefined behavior (such as
52092              another thread closing an fd it knows nothing about), cleanup
52093              from any defined failures would require stopping and reaping the
52094              child process, which may have worse consequences. */
52095           posix_spawn_file_actions_destroy(&actions);
52096           close(fds[!target]);
52097           popen_add_pair(stream, pid);
52098           popen_unlock();
52099           if (mode[1] != 'e') {
52100             int flags = fcntl(fds[target], F_GETFD);
52101             if (flags >= 0)
52102               fcntl(fds[target], F_SETFD, flags & ~FD_CLOEXEC);
52103           }
52104           return stream;
52105         }
```

52106 Both examples can share a common *pclose*( ) implementation.

```
52107         int pclose(FILE *stream)
52108         {
52109           int status;
52110           popen_lock();
52111           pid_t pid = popen_get_pid(stream);
52112           if (pid < 0) {
52113             popen_unlock();
52114             return -1;
```



```
    }
    popen_remove(stream);
    popen_unlock();
    fclose(stream); /* Ignore failure */
    while (waitpid(pid, &status, 0) == -1) {
      if (errno != EINTR) {
        status = -1;
        break;
      }
    }
    return status;
}
```

Note that, while a particular implementation of *popen*( ) (such as the two above) can assume a particular path for the shell, such a path is not necessarily valid on another system. The above examples are not portable, and are not intended to be.

Earlier versions of this standard required the *command* string to be passed as the next argument after `"-c"` (omitting `"--"`). This meant that portable applications needed to take care not to pass a command string beginning with <hyphen-minus> (`'-'`) or <plus-sign> (`'+'`), as it would then be interpreted as containing options. Now that implementations are required to pass the `"--"`, applications no longer need to do this.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fork*( ), *pclose*( ), *pipe*( ), *sysconf*( ), *system*( ), *wait*( ), *waitid*( )

XBD **<stdio.h>**

XCU *sh*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

A statement is added to the DESCRIPTION indicating that pipe streams are byte-oriented.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The optional [EMFILE] error condition is added.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/67 is applied, adding the example to the EXAMPLES section.

Issue 7

Austin Group Interpretation 1003.1-2001 #029 is applied, clarifying the values for *mode* in the DESCRIPTION.

SD5-XSH-ERN-149 is applied, changing the {STREAM_MAX} [EMFILE] error condition from a ``may fail'' to a ``shall fail''.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0432 [14] is applied.

Issue 8

Austin Group Defects 411 and 1318 are applied, adding the `'e'` mode modifier and FD_CLOFORK, and adding example implementations of *popen*( ) and *pclose*( ) in the RATIONALE section.

Austin Group Defect 1317 is applied, making it implementation-defined whether the handlers registered with *pthread_atfork*( ) are called.

Austin Group Defect 1440 is applied, adding a `"--"` argument to the specified *execl*( ) call.

Austin Group Defect 1526 is applied, making typographic changes relating to *mode* values for consistency with *fopen*( ).

52166 **NAME**

52167 posix_close — close a file descriptor

52168 **SYNOPSIS**

52169 `#include <unistd.h>`

52170 `int posix_close(int fildes, int flag);`

52171 **DESCRIPTION**

52172 Refer to *close( )*.

NAME

posix_devctl — device control

SYNOPSIS

DC

```
#include <devctl.h>

int posix_devctl(int fildes, int dcmd, void *restrict dev_data_ptr,
    size_t nbyte, int *restrict dev_info_ptr);
```

DESCRIPTION

The *posix_devctl*( ) function shall cause the device control command *dcmd* to be passed to the driver identified by *fildes*. Associated data shall be passed to and/or from the driver depending on direction information encoded in the *dcmd* argument or as implied in the *dcmd* argument by the design and implementation of the driver.

If the *dev_data_ptr* argument is not a null pointer, it shall be a pointer to a buffer that is provided by the caller and that contains data bytes to be passed to the driver or provides space for receiving data bytes to be passed back from the driver, or both.

If the data bytes are to be passed to the driver, at least *nbyte* bytes of associated data shall be made available to the driver; if the data bytes are to be passed from the driver, no more than *nbyte* bytes shall be passed.

The driver may be executing in an address space different from the address space of the calling thread. Therefore, if the data bytes passed to the driver (i.e., the contents of the memory area starting at *dev_data_ptr* and continuing for *nbyte* bytes) contain pointers to memory in the address space of the calling thread and the driver uses these pointers to access that memory, the effects are unspecified.

OB If *dev_data_ptr* is not a null pointer and *nbyte* is zero, the amount of data passed to and/or from the driver is unspecified. This feature is obsolescent and is only provided for compatibility with existing device drivers.

If *dev_data_ptr* is a null pointer, there shall be no data bytes passed between the caller and the driver other than the data specified in the rest of the arguments to *posix_devctl*( ) and in its return value.

The *dev_info_ptr* argument provides the opportunity to return an integer number containing additional device information, instead of just a success/failure indication. For implementation-provided *dcmd* values, it is implementation-defined whether each such value causes the **int** pointed to by *dev_info_ptr* to be set and, if set, what value it is set to.

For each supported device, the set of valid *dcmd* commands, the associated data interpretation, and the effects of the command on the device are all defined by the driver for the device identified by *fildes*, and are therefore implementation-defined for implementation-provided device drivers.

RETURN VALUE

Upon successful completion, *posix_devctl*( ) shall return zero; otherwise an error number shall be returned to indicate the error. The value returned in the **int** value pointed to by *dev_info_ptr* is driver dependent.

ERRORS

The *posix_devctl*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid open file descriptor.

The *posix_devctl*( ) function may fail if:

52217 [EINTR] The *posix_devctl*( ) function was interrupted by a signal.

52218 [EINVAL] The *nbyte* argument exceeds an implementation-defined maximum or is less
52219 than the minimum number of bytes required for this command.

52220 [EINVAL] The *dcmd* argument is not valid for this device.

52221 [ENOTTY] The *fildes* argument is not associated with a character special file that accepts
52222 control functions.

52223 [EPERM] The requesting process does not have the appropriate privilege to request the
52224 device to perform the specified command.

52225 Driver code may detect other errors, but the error numbers returned are driver dependent. See
52226 ``Recommended Practice for Driver-Detected Errors'' in RATIONALE.

52227 If the *posix_devctl*( ) function fails, the effect of this failed function on the device is driver
52228 dependent. Corresponding data might be transferred, partially transferred, or not transferred at
52229 all.

52230 **EXAMPLES**

52231 None.

52232 **APPLICATION USAGE**

52233 None.

52234 **RATIONALE**

52235 **Background**

52236 An interface to be included in the POSIX standard should improve source code portability of
52237 application programs. In traditional UNIX practice, *ioctl*( ) was used to handle special devices.
52238 Therefore, a general specification of its arguments cannot be written. Based on this fact, in the
52239 past many people claimed that *ioctl*( ), or something close to it, had no place in the POSIX
52240 standards.

52241 Against this perception stood the widespread use of *ioctl*( ) to interface to all sorts of drivers for a
52242 vast variety of hardware used in all areas of general-purpose, realtime, and embedded
52243 computing, such as analog-digital converters, counters, and video graphic devices. These
52244 devices provide a set of services that cannot be represented or used in terms of *read*( ) or *write*( )
52245 calls.

52246 The arguments in favor of *ioctl*( ) standardization can be summarized as follows:

52247 Even if *ioctl*( ) addresses very different hardware, many of these devices either are actually the
52248 same, interfaced to different computer systems with different implementations of operating
52249 systems, or belong to classes of devices with rather high commonality in their functions, e.g.,
52250 analog-digital converters or digital-analog converters. Growing standardization of the control
52251 and status register (CSR) space of these devices allows exploitation of a growing similarity of
52252 control codes and data for these devices. A general mechanism is needed to control these
52253 devices.

52254 In all these cases, a standardized interface from the application program to drivers for these
52255 devices will improve source code portability.

52256 Even if control codes and device data have to be changed when porting applications from one
52257 system to another, the definition of *ioctl*( ) largely improves readability of a program handling
52258 special devices. Changes are confined to more clearly labeled places.

52259 A driver for a specific device normally cannot be considered portable *per se*, but an application

that uses this driver can be made portable if all interfaces needed are well defined and standardized. Users and integrators of realtime systems often add device drivers for specific devices, and a standard interface simplifies this process. Also, device drivers often follow their special hardware from system to system.

In recognition of these reasons, The Open Group included *ioctl*( ) in the The Single UNIX Specification, Version 1, and the interface was later incorporated into POSIX.1 under the XSI STREAMS option (although that option was subsequently removed).

The *posix_devctl*( ) interface defined in this standard provides an alternative to the the various *ioctl*( ) implementations with a standard interface that captures the extensibility of *ioctl*( ), but avoids several of its deficiencies, which is mentioned in ``Relationship to *ioctl*( ) and the Perceived Needs for Improvement'' below.

Existing Practice

The *ioctl*( ) interface is widely used. It has provided the generality mentioned above. Existing practice encodes into the second parameter information about data size and direction in some systems. An example of such an encoding is the use in BSD 4.3 of two bits of the command word as read/write bits. However, *ioctl*( ) has definite problems with the way that its sometimes optional third parameter can be interpreted.

This practice is similar to the existing POSIX *fcntl*( ) function, in which the third parameter can be optional for the F_GETFD and F_GETFL commands, an **int** when used with the F_DUPFD, F_SETFD, or F_SETFL commands, or a **struct flock** when used with the F_GETLK, F_SETLK, or F_SETLKW commands. However, the *fcntl*( ) interface defines two distinct and known data types as possible for the third parameter. This is not the case in the *ioctl*( ) interface, where any number of device driver specific structures and commands are used.

Relationship to *ioctl*( ) and the Perceived Needs for Improvement

Section A.11 (on page 3718) briefly mentions some of the perceived deficiencies in existing implementations of the *ioctl*( ) function, in the context of those *ioctl*( ) commands used to implement terminal control. The standard developers decided that, since the set of such control operations was fairly well defined, suitable encapsulations such as *tcsetattr*( ), *tcsendbreak*( ), and *tcdrain*( ) could be standardized. These interfaces, while successfully standardizing portable terminal control operations, are not extensible to arbitrary user-supplied devices.

There are several perceived deficiencies with the *ioctl*( ) function that drove the development of the *posix_devctl*( ) interface as an alternative:

- The major problem with *ioctl*( ) is that the third argument (when one is passed) varies in both size and type according to the second (command) argument. It is not unprecedented in POSIX, or standards in general, for a function to accept a generic pointer; consider the ISO C function *fread*( ), or the POSIX functions *read*( ) and *mmap*( ). However, in all such instances, the generic pointer is accompanied by a size argument that specifies the size of the pointed-to object. Unlike the Ada language, it is, and has always been, the C programmer's responsibility to ensure that these two arguments form a consistent specification of the passed object. But traditional *ioctl*( ) implementations do not allow the user to specify the size of the pointed-to object; that size is instead fixed implicitly by the specified command (passed as another argument). The *posix_devctl*( ) interface improves upon *ioctl*( ) in that it allows the user to specify the object size, thereby restoring the familiar C paradigm for passing a generic object by pointer/size pair.

- A secondary problem with *ioctl*( ) is that the third argument is sometimes permitted to be interpreted as an integer (**int**). The *posix_devctl*( ) interface clearly requires the *dev_data_ptr* argument to be a pointer.
- A related problem with *ioctl*( ) is that the direction(s) in which data are transferred to or from the pointed-to object is neither specified explicitly as an argument (as with *mmap*( )), nor implied by the *ioctl*( ) function (as with *read*( )/*write*( ), *fread*( )/*fwrite*( ), or *fgets*( )/*fputs*( )). Instead, the direction is implied by the command argument. In traditional implementations, only the device driver knows the interpretation of the commands and whether data bytes are to be transferred to or from the pointed-to object. But in networked implementations, generic portions of the operating system may need to know the direction to ensure that data bytes are passed properly between a client and a server, separately from device driver concerns. Two implementation-specific solutions to this problem are to always assume data bytes need to be transferred in both directions, or to encode the implied direction into the command word along with the fixed data size. The *posix_devctl*( ) interface already provides the implementation with an explicit size parameter. Since the direction is already known implicitly to both the application and the driver and since workable methods exist for implementations to ascertain that direction if required, this perceived problem is strictly an implementation issue and solvable without further impact on the interface.
- Finally, *posix_devctl*( ) improves upon *ioctl*( ) by adopting the new style of error return, avoiding all the problems *errno* brings to multi-threaded applications. Because the driver-specific information carried by the non-error return values of *ioctl*( ) still potentially needs to be passed to the application, *posix_devctl*( ) adds the *dev_info_ptr* argument to specify where this information should be stored.

Which Differences Between *posix_devctl*( ) and *ioctl*( ) Are Acceptable?

Any differences between the definitions of *posix_devctl*( ) and *ioctl*( ) have to be perceived as a clear improvement by the community of potential users. Drivers for normal peripherals are typically written by highly specialized professionals. Drivers for the special devices are very often written by the application developer or by the hardware designer. Any interface definition that can be seen as overly complicated will simply not be accepted.

Nevertheless, a few simple and useful improvements to *ioctl*( ) are possible, specifically the improvement of type checking, and justify the definition of a new interface.

The major difference between the two interfaces is the addition of the size of the device data. For enhanced compatibility with existing *ioctl*( ) implementations, this size can be specified as zero; in this case the amount of data passed is unspecified. (This allows a macro definition of *ioctl*( ) that converts it into a *posix_devctl*( ) call.) In any case, the data size argument does not contradict the general goal of being able to implement *posix_devctl*( ) using the existing *ioctl*( ) interfaces provided in current UNIX systems and other POSIX implementations because the standard allows but does not require checking the size of the device data. Although the third argument of the *ioctl*( ) function does not specify a size, it is implicit in the specific combination of control command and driver and, therefore, known to the driver implementation.

The method of indicating error return values differs from traditional *ioctl*( ) implementations, but it does not preclude the construction of *posix_devctl*( ) as a macro built upon *ioctl*( ), which was one of the original design goals.

Rationale for the *dev_info_ptr* Argument

The POSIX.26 developers felt that it was important to preserve the current *ioctl*( ) functionality of allowing a device driver to return some arbitrary piece of information instead of just a success/failure indication. Such information might be, for example, the number of bytes received, the number of bytes that would not fit into the buffer pointed at by *dev_data_ptr*, the data type indication, or the device status. Current practice for device drivers and *ioctl*( ) usage allows such a device-dependent return value. Thus, the concept of an additional output argument, *dev_info_ptr*, was born.

Rationale for No *direction* Argument

The initial specification for *posix_devctl*( ) contained an additional argument that specified the direction of data flow, i.e., to the driver and/or from the driver. This argument was later removed for the following reasons:

- The argument was redundant. Most (if not all) existing implementations encode the direction data either explicitly or implicitly in the command word.
- The argument increased the probability of programming errors, since it must be made to agree with the direction information already encoded or implied in the command word or an error would occur.
- The only real use of the argument would be if new drivers were written that supported generic commands such as TRANSFER_CONTROL_DATA, which was modified by the direction argument to indicate in which direction the data should be transferred. This is contrary to current practice that uses command pairs such as GET_CONTROL_DATA and PUT_CONTROL_DATA.
- The primary purpose of the direction argument was to allow higher levels of the system to identify the direction of data transfers, particularly in the case of remote devices, without having to understand all the commands of all the devices on the system. Implementations that need to ascertain the direction of data transfer from a command word will define a consistent convention for encoding the direction into each command word, and all device drivers supplied by the user must adhere to this convention.

Thus, the data direction argument was removed.

Rationale for Not Defining the Direction Encoding in the *dcmd* Argument

The POSIX.26 developers gave consideration to defining the direction encoding in the *dcmd* argument, but decided against doing so. No particular benefit was seen to a predefined encoding, as long as the encoding was used consistently across the entire implementation and was well known to the implementation.

In addition, although only one encoding (BSD's) employed for *ioctl*( ) was known among the members of the small working group, it could not be ruled out that other encodings already existed, and no reason for precluding these encodings was seen.

Finally, system or architectural constraints might make a chosen standard encoding difficult to use on a given implementation.

Thus, this standard does not define a direction encoding. Specifying a standard encoding is actually a small part of a larger and more contentious objective, that of specifying a complete set of interfaces for portable device drivers. If a future POSIX standard specifies such interfaces, the issue of device control direction encoding will necessarily be addressed as part of that specification.

Recommended Practice for Handling Data Size Errors

In the event that the amount of data from the device is too large to fit into the specified buffer, as much data as will fit should be transferred, and the error posted. The retained data will aid in debugging, even if some of the data is lost.

Recommended Practice for *nbyte* == 0

The feature that permits an unspecified amount of control data to be transferred if *nbyte* is zero exists only for compatibility with existing device driver usage of *ioctl*( ), i.e., when *ioctl*( ) is implemented on top of *posix_devctl*( ) and the device driver transfers an amount of data implied by the command.

Implementations in which *posix_devctl*( ) is built as a library routine on top of *ioctl*( ) may not be able to make checks on the *nbyte* argument. However, newly developed applications using *posix_devctl*( ) should always use an appropriate value for the *nbyte* argument, for portability to implementations directly supporting *posix_devctl*( ) in which the device drivers may be able to honor the application's *nbyte* argument or return the error [EINVAL] if the argument is an unacceptable value. Device drivers designed for those systems should interpret a zero value of *nbyte* as no data to be transferred.

Recommended Practice for Driver-Detected Errors

If the driver detects the following error conditions, it is recommended that the *posix_devctl*( ) function fail and return the corresponding error number:

[EBUSY] The control operation could not complete successfully because the device was in use by another process, or the driver was unable to carry out the request due to an outstanding operation in progress.

[EINVAL] The arguments *dev_data_ptr* and *nbyte* define a buffer too small to hold the amount of data expected by or to be returned by this driver.

[EIO] The control operation could not complete successfully because the driver detected a hardware error.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD Section 4.21 (on page 108), **<devctl.h>**

CHANGE HISTORY

First released in Issue 8. Derived from POSIX.26.

NAME

posix_fadvise — file advisory information (**ADVANCED REALTIME**)

SYNOPSIS

ADV

```
#include <fcntl.h>

int posix_fadvise(int fd, off_t offset, off_t len, int advice);
```

DESCRIPTION

The *posix_fadvise*( ) function shall advise the implementation on the expected behavior of the application with respect to the data in the file associated with the open file descriptor, *fd*, starting at *offset* and continuing for *len* bytes. The specified range need not currently exist in the file. If *len* is zero, all data from *offset* to the largest possible value of the file offset for that file shall be specified. The implementation may use this information to optimize handling of the specified data. The *posix_fadvise*( ) function shall have no effect on the semantics of other operations on the specified data, although it may affect the performance of other operations.

The advice to be applied to the data is specified by the *advice* parameter and may be one of the following values:

POSIX_FADV_NORMAL
: Specifies that the application has no advice to give on its behavior with respect to the specified data. It is the default characteristic if no advice is given for an open file.

POSIX_FADV_SEQUENTIAL
: Specifies that the application expects to access the specified data sequentially from lower offsets to higher offsets.

POSIX_FADV_RANDOM
: Specifies that the application expects to access the specified data in a random order.

POSIX_FADV_WILLNEED
: Specifies that the application expects to access the specified data in the near future.

POSIX_FADV_DONTNEED
: Specifies that the application expects that it will not access the specified data in the near future.

POSIX_FADV_NOREUSE
: Specifies that the application expects to access the specified data once and then not reuse it thereafter.

These values are defined in **<fcntl.h>**.

RETURN VALUE

Upon successful completion, *posix_fadvise*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_fadvise*( ) function shall fail if:

[EBADF] The *fd* argument is not a valid file descriptor.

[EINVAL] The value of *advice* is invalid, or the value of *len* is less than zero.

[ESPIPE] The *fd* argument is associated with a pipe or FIFO.

52464 **EXAMPLES**

52465 None.

52466 **APPLICATION USAGE**

52467 The *posix_fadvise*( ) function is part of the Advisory Information option and need not be
52468 provided on all implementations.

52469 **RATIONALE**

52470 None.

52471 **FUTURE DIRECTIONS**

52472 None.

52473 **SEE ALSO**

52474 *posix_madvise*( )

52475 XBD **<fcntl.h>**

52476 **CHANGE HISTORY**

52477 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

52478 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

52479 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/68 is applied, changing the function
52480 prototype in the SYNOPSIS section. The previous prototype was not large file-aware, and the
52481 standard developers felt it acceptable to make this change before implementations of this
52482 function become widespread.

52483 **Issue 7**

52484 Austin Group Interpretation 1003.1-2001 #024 is applied, changing the definition of the
52485 [EINVAL] error.

52486 **Issue 8**

52487 Austin Group Defect 1135 is applied, clarifying the requirements when *len* is zero.

NAME

posix_fallocate — file space control (**ADVANCED REALTIME**)

SYNOPSIS

ADV
```
#include <fcntl.h>

int posix_fallocate(int fd, off_t offset, off_t len);
```

DESCRIPTION

The *posix_fallocate*( ) function shall ensure that any required storage for regular file data starting at *offset* and continuing for *len* bytes is allocated on the file system storage media. If *posix_fallocate*( ) returns successfully, subsequent writes to the specified file data shall not fail due to the lack of free space on the file system storage media.

If the *offset+len* is beyond the current file size, then *posix_fallocate*( ) shall adjust the file size to *offset+len*. Otherwise, the file size shall not be changed.

It is implementation-defined whether a previous *posix_fadvise*( ) call influences allocation strategy.

Space allocated via *posix_fallocate*( ) shall be freed by a successful call to *creat*( ) or *open*( ) that truncates the size of the file. Space allocated via *posix_fallocate*( ) may be freed by a successful call to *ftruncate*( ) that reduces the file size to a size smaller than *offset+len*.

RETURN VALUE

Upon successful completion, *posix_fallocate*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_fallocate*( ) function shall fail if:

[EBADF] The *fd* argument is not a valid file descriptor.

[EBADF] The *fd* argument references a file that was opened without write permission.

[EFBIG] The value of *offset+len* is greater than the maximum file size.

XSI [EFBIG] The value of *offset+len* exceeds the file size limit of the process. A SIGXFSZ signal shall also be generated for the thread.

[EINTR] A signal was caught during execution.

[EINVAL] The *len* argument is less than zero, or the *offset* argument is less than zero.

[EIO] An I/O error occurred while reading from or writing to a file system.

[ENODEV] The *fd* argument does not refer to a regular file.

[ENOSPC] There is insufficient free space remaining on the file system storage media.

[ENOTSUP] The underlying file system does not support this operation.

[ESPIPE] The *fd* argument is associated with a pipe or FIFO.

The *posix_fallocate*( ) function may fail if:

[EINVAL] The *len* argument is zero.

52524 **EXAMPLES**

52525 None.

52526 **APPLICATION USAGE**

52527 The *posix_fallocate*( ) function is part of the Advisory Information option and need not be
52528 provided on all implementations.

52529 Not all file systems are capable of supporting *posix_fallocate*( ), in which case the function will
52530 return [ENOTSUP]. However, if a system supports the Advisory Information option, there must
52531 be at least one file system that is capable of supporting this function and is available for use in
52532 conforming environments. The *pathconf*( ) and *fpathconf*( ) functions can be used to determine
52533 whether a file in a particular file system supports *posix_fallocate*( ).

52534 **RATIONALE**

52535 None.

52536 **FUTURE DIRECTIONS**

52537 None.

52538 **SEE ALSO**

52539 *creat*( ), *ftruncate*( ), *open*( ), *unlink*( )

52540 XBD **<fcntl.h>**

52541 **CHANGE HISTORY**

52542 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

52543 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

52544 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/69 is applied, changing the function
52545 prototype in the SYNOPSIS section. The previous prototype was not large file-aware, and the
52546 standard developers felt it acceptable to make this change before implementations of this
52547 function become widespread.

52548 **Issue 7**

52549 Austin Group Interpretations 1003.1-2001 #022, #024, and #162 are applied, changing the
52550 definition of the [EINVAL] error.

52551 **Issue 8**

52552 Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

52553 Austin Group Defect 684 is applied, requiring an [ENOTSUP] error instead of [EINVAL] when
52554 the underlying file system does not support *posix_fallocate*( ).

52555 Austin Group Defect 687 is applied, changing the APPLICATION USAGE section.

52556 Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error
52557 relating to the file size limit for the process.

52558 **NAME**

52559 posix_getdents — read directory entries

52560 **SYNOPSIS**

52561 ```
#include <dirent.h>
```

52562 ```
ssize_t posix_getdents(int fildes, void *buf, size_t nbyte, int flags);
```

52563 **DESCRIPTION**

52564 The *posix_getdents*( ) function shall attempt to read directory entries from the directory associated
52565 with the open file descriptor *fildes* and shall place information about the directory entries and the
52566 files they refer to in **posix_dent** structures in the buffer pointed to by *buf*, up to a maximum of
52567 *nbyte* bytes. The number of **posix_dent** structures populated in *buf* may be fewer than the
52568 number that will fit in *nbyte* bytes, but shall be at least one if *nbyte* is greater than the size of the
52569 **posix_dent** structure plus {NAME_MAX} and *fildes* is not currently at end-of-file.

52570 The application shall ensure that *buf* is aligned suitably to point to a **posix_dent** structure. The
52571 alignment needed shall not be more restrictive than the alignment provided by *malloc*( ). Strictly
52572 conforming applications shall ensure that the value of *flags* is zero; other applications can set it to
52573 a value constructed by a bitwise-inclusive OR of implementation-defined bitwise-distinct flag
52574 values.

52575 Each **posix_dent** structure returned in *buf* shall be located at an address that satisfies the
52576 implementation's alignment requirements for the **posix_dent** structure and shall be populated
52577 as follows:

52578 • The value of the *d_ino* member shall be set to the file serial number of the file named by the
52579 *d_name* member.

52580 • The value of the *d_reclen* member shall be set to the number of bytes occupied by this entry
52581 in *buf*, including any padding bytes needed before the next entry, if any. If this is the last
52582 entry in *buf*, *d_reclen* shall include any padding bytes needed to make the address of this
52583 entry plus *d_reclen* bytes satisfy the alignment requirements for the **posix_dent** structure.

52584 • The value of the *d_type* member shall be set to indicate the file type of the named file, if the
52585 file type can be determined without needing to use the file serial number to obtain the
52586 file's metadata; otherwise it may be set to DT_UNKNOWN. If the file type is determined
52587 and it is one of the file types defined in this standard, the value of *d_type* shall be DT_BLK,
52588 DT_CHR, DT_DIR, DT_FIFO, DT_LNK, DT_REG, DT_SOCK, DT_MQ, DT_SEM,
52589 TYM DT_SHM, or DT_TMO (see **<dirent.h>**). If it is determined but is not a standard file type,
52590 the value of *d_type* shall not equal any of those listed here.

52591 • The *d_name* member shall be a filename string, and (if not dot or dot-dot) shall contain the
52592 same byte sequence as the last pathname component of the string used to create the
52593 directory entry, plus the terminating NUL byte.

52594 If the *d_name* member names a symbolic link, the values of the *d_ino* and *d_type* members shall
52595 be set to the values for the symbolic link itself.

52596 The *posix_getdents*( ) function shall start reading at the current file offset in the open file
52597 description associated with *fildes*. On successful return, the file offset shall be incremented to
52598 point to the directory entry immediately following the last entry whose information was
52599 returned in *buf*, or to point to end-of-file if there are no more directory entries. On failure, the
52600 value of the file offset is unspecified. The current file offset can be set and retrieved using *lseek*( )
52601 on the open file description associated with *fildes*. The behavior is unspecified if *lseek*( ) is used
52602 to set the file offset to a value other than zero or a value returned by a previous call to *lseek*( ) on
52603 the same open file description.

The *posix_getdents*( ) function shall not return directory entries containing empty names. If entries for dot or dot-dot exist, a sequence of calls that reads from offset zero to end-of-file shall return one entry for dot and one entry for dot-dot; otherwise, they shall not be returned.

Upon successful completion, *posix_getdents*( ) shall mark for update the last data access timestamp of the directory.

If *fildes* is a file descriptor associated with a directory stream opened using *fdopendir*( ) or *opendir*( ), the behavior is unspecified.

If *posix_getdents*( ) is called concurrently with an operation that adds, deletes, or modifies a directory entry, the results from *posix_getdents*( ) shall reflect either all of the effects of the concurrent operation or none of them. If a sequence of calls to *posix_getdents*( ) is made that reads from offset zero to end-of-file and a file is removed from or added to the directory between the first and last of those calls, whether the sequence of calls returns an entry for that file is unspecified.

RETURN VALUE

Upon successful completion, either a non-negative integer shall be returned indicating the number of bytes occupied by the **posix_dent** structures placed in *buf* or 0 shall be returned indicating the end of the directory was reached without any directory entries being placed in *buf*. Otherwise, –1 shall be returned and *errno* shall be set to indicate the error.

ERRORS

The *posix_getdents*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid file descriptor open for reading.

[EINVAL] The *nbyte* argument is not large enough to contain the information to be returned about the directory entry located at the current file offset.

[ENOENT] The current file offset is not located at a valid directory entry.

[ENOTDIR] The *fildes* argument is associated with a non-directory file.

[EOVERFLOW] One of the values in a structure to be placed in *buf* cannot be represented correctly.

The *posix_getdents*( ) function may fail if:

[EIO] A physical I/O error has occurred.

[ENOMEM] Insufficient memory was available to fulfill the request.

EXAMPLES

This example function lists the files in a specified directory with their file serial number and file type. If the file type is not available from *posix_getdents*( ), it is obtained using *fstatat*( ).

```
#include <dirent.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <unistd.h>

#define ENTBUFSIZ 10240

int list_dir(const char *dirnam)
{
    int fd = open(dirnam, O_RDONLY | O_DIRECTORY);
```



```
52647           if (fd == -1)
52648               return -1;
52649           char *buf = malloc(ENTBUFSIZ);
52650           if (buf == NULL)
52651           {
52652               close(fd);
52653               return -1;
52654           }
52655           ssize_t bytesinbuf;
52656           for(;;)
52657           {
52658               ssize_t nextent = 0;
52659               bytesinbuf = posix_getdents(fd, buf, ENTBUFSIZ, 0);
52660               if (bytesinbuf <= 0)
52661                   break;
52662               do {
52663                   const char *ftype;
52664                   struct posix_dent *entp = (void *)&buf[nextent];
52665                   if (entp->d_type == DT_UNKNOWN)
52666                   {
52667                       struct stat stbuf;
52668                       if (fstatat(fd, entp->d_name, &stbuf,
52669                                   AT_SYMLINK_NOFOLLOW) == -1)
52670                           ftype = "?";
52671                       else
52672                           ftype = S_ISBLK(stbuf.st_mode) ? "b" :
52673                                S_ISCHR(stbuf.st_mode) ? "c" :
52674                                S_ISDIR(stbuf.st_mode) ? "d" :
52675                                S_ISFIFO(stbuf.st_mode) ? "p" :
52676                                S_ISLNK(stbuf.st_mode) ? "l" :
52677                                S_ISREG(stbuf.st_mode) ? "r" :
52678                                S_ISSOCK(stbuf.st_mode) ? "s" :
52679                                S_TYPEISMQ(&stbuf) ? "mq" :
52680                                S_TYPEISSEM(&stbuf) ? "sem" :
52681                                S_TYPEISSHM(&stbuf) ? "shm" :
52682       #ifdef S_TYPEISTMO
52683                                S_TYPEISTMO(&stbuf) ? "tmo" :
52684       #endif
52685                                "?";
52686                   }
52687                   else
52688                   {
52689                       ftype = entp->d_type == DT_BLK ? "b" :
52690                               entp->d_type == DT_CHR ? "c" :
52691                               entp->d_type == DT_DIR ? "d" :
52692                               entp->d_type == DT_FIFO ? "p" :
52693                               entp->d_type == DT_LNK ? "l" :
52694                               entp->d_type == DT_REG ? "r" :
52695                               entp->d_type == DT_SOCK ? "s" :
52696                               entp->d_type == DT_MQ ? "mq" :
```



```
                                entp->d_type == DT_SEM ? "sem" :
                                entp->d_type == DT_SHM ? "shm" :
#ifdef DT_TMO
                                entp->d_type == DT_TMO ? "tmo" :
#endif
                                "?";
                    }

                    printf("%ld\t%s\t%s\n", (long)entp->d_ino, ftype,
                        entp->d_name);

                    nextent += entp->d_reclen;

                } while (nextent < bytesinbuf);
            }

            close(fd);
            free(buf);
            return bytesinbuf;
        }
```

APPLICATION USAGE

If an array of **posix_dent** structures (which is only possible on implementations where *d_name* is not a flexible array member) is used to provide the storage for *buf* in order to satisfy the alignment requirement, it should be noted that the number of array elements used to size the array may bear little or no relation to the number of directory entries that can be stored in it. It is recommended that the number of elements is calculated from the desired size in bytes, for example:

```
#define DESIREDSIZE 10240
struct posix_dent buf[DESIREDSIZE / sizeof(struct posix_dent) + 1];
size_t nbyte = sizeof buf;
```

When *posix_getdents*( ) is called with a *buf* that is not type **char** *, it is important to note that *d_reclen* is a byte count and therefore any pointer arithmetic involved in calculating the start of the next entry needs to use a **char** * pointer.

On implementations where directory entries in a directory take up more space than the corresponding **posix_dent** structures in *buf*, a call to *posix_getdents*( ) may read *nbyte* bytes from the directory, resulting (in most cases) in the actual number of bytes placed in *buf* being less than *nbyte*.

One advantage of *posix_getdents*( ) is that it provides the file type of each directory entry (if available), whereas *readdir*( ) only does so on implementations that have the file type as a non-standard additional member of the **dirent** structure. Knowing the file type can greatly reduce the number of *fstatat*( ) calls that need to be made when traversing the file hierarchy.

Whether or not a file's type can be determined without needing to use the file serial number to obtain the file's metadata may vary across the different file system types supported by an implementation. Therefore applications should not assume that if *d_type* contains known file types (i.e. not DT_UNKNOWN) for entries in a given directory then it will also contain known file types for entries in subdirectories of that directory or in its parent.

Since the *d_reclen* value for the last entry in *buf* includes padding to satisfy alignment requirements, applications can grow the buffer and call *posix_getdents*( ) again to append to it without needing to perform an alignment calculation.

RATIONALE

The *posix_getdents*( ) function was derived from existing *getdents*( ) functions but the name was changed because the existing *getdents*( ) functions differed in various ways, in particular the type of the second argument (structure pointer or **void** *), the members of the populated structures, and the error numbers used for some conditions. The name change also provided an opportunity to add a *flags* argument to provide for future extensibility.

Implementations are encouraged to include support for a DT_FORCE_TYPE flag which, when that bit is set in *flags*, causes *posix_getdents*( ) to look up the file type if it can not be obtained from the directory entry. This will allow applications that need to know the file type of every directory entry to keep the cost of these lookups to the minimum needed to obtain the type at the file system level, without the additional overhead of making a call to *fstatat*( ) for every file (that has *d_type* equal to DT_UNKNOWN).

Some existing *getdents*( ) or similar functions return directory entry structures for deleted directory entries in *buf*, marked with a special value of one of the structure members to distinguish them from non-deleted entries. This behavior is not allowed for *posix_getdents*( ), although the data from a deleted directory entry may be present in *buf* in the form of extra padding on the end of the previous entry.

FUTURE DIRECTIONS

A future version of this standard may add a DT_FORCE_TYPE flag as described in RATIONALE.

SEE ALSO

fdopendir( ), *fstatat*( ), *lseek*( ), *readdir*( )

XBD **<dirent.h>**

CHANGE HISTORY

First released in Issue 8.

NAME

posix_madvise — memory advisory information and alignment control (**ADVANCED REALTIME**)

SYNOPSIS

ADV

```
#include <sys/mman.h>

int posix_madvise(void *addr, size_t len, int advice);
```

DESCRIPTION

The *posix_madvise*( ) function shall advise the implementation on the expected behavior of the application with respect to the data in the memory starting at address *addr*, and continuing for *len* bytes. The implementation may use this information to optimize handling of the specified data. The *posix_madvise*( ) function shall have no effect on the semantics of access to memory in the specified range, although it may affect the performance of access.

The implementation may require that *addr* be a multiple of the page size, which is the value returned by *sysconf*( ) when the name value _SC_PAGESIZE is used.

The advice to be applied to the memory range is specified by the *advice* parameter and may be one of the following values:

POSIX_MADV_NORMAL
: Specifies that the application has no advice to give on its behavior with respect to the specified range. It is the default characteristic if no advice is given for a range of memory.

POSIX_MADV_SEQUENTIAL
: Specifies that the application expects to access the specified range sequentially from lower addresses to higher addresses.

POSIX_MADV_RANDOM
: Specifies that the application expects to access the specified range in a random order.

POSIX_MADV_WILLNEED
: Specifies that the application expects to access the specified range in the near future.

POSIX_MADV_DONTNEED
: Specifies that the application expects that it will not access the specified range in the near future.

These values are defined in the **<sys/mman.h>** header.

RETURN VALUE

Upon successful completion, *posix_madvise*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_madvise*( ) function shall fail if:

[EINVAL] The value of *advice* is invalid.

[ENOMEM] Addresses in the range starting at *addr* and continuing for *len* bytes are partly or completely outside the range allowed for the address space of the calling process.

The *posix_madvise*( ) function may fail if:

[EINVAL] The value of *addr* is not a multiple of the value returned by *sysconf*( ) when the name value _SC_PAGESIZE is used.

[EINVAL] The value of *len* is zero.

EXAMPLES

None.

APPLICATION USAGE

The *posix_madvise*( ) function is part of the Advisory Information option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mmap( ), *posix_fadvise*( ), *sysconf*( )

XBD **<sys/mman.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #102 is applied.

52825 **NAME**

52826 posix_mem_offset — find offset and length of a mapped typed memory block (**ADVANCED**
52827 **REALTIME**)

52828 **SYNOPSIS**

52829 TYM
```
#include <sys/mman.h>

int posix_mem_offset(const void *restrict addr, size_t len,
    off_t *restrict off, size_t *restrict contig_len,
    int *restrict fildes);
```

52833 **DESCRIPTION**

52834 The *posix_mem_offset*( ) function shall return in the variable pointed to by *off* a value that
52835 identifies the offset (or location), within a memory object, of the memory block currently
52836 mapped at *addr*. The function shall return in the variable pointed to by *fildes*, the descriptor used
52837 (via *mmap*( )) to establish the mapping which contains *addr*. If that descriptor was closed since
52838 the mapping was established, the returned value of *fildes* shall be –1. The *len* argument specifies
52839 the length of the block of the memory object the user wishes the offset for; upon return, the
52840 value pointed to by *contig_len* shall equal either *len*, or the length of the largest contiguous block
52841 of the memory object that is currently mapped to the calling process starting at *addr*, whichever
52842 is smaller.

52843 If the memory object mapped at *addr* is a typed memory object, then if the *off* and *contig_len*
52844 values obtained by calling *posix_mem_offset*( ) are used in a call to *mmap*( ) with a file descriptor
52845 that refers to the same memory pool as *fildes* (either through the same port or through a different
52846 port), and that was opened with neither the POSIX_TYPED_MEM_ALLOCATE nor the
52847 POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, the typed memory area that is mapped shall
52848 be exactly the same area that was mapped at *addr* in the address space of the process that called
52849 *posix_mem_offset*( ).

52850 If the memory object specified by *fildes* is not a typed memory object, then the behavior of this
52851 function is implementation-defined.

52852 **RETURN VALUE**

52853 Upon successful completion, the *posix_mem_offset*( ) function shall return zero; otherwise, the
52854 corresponding error status value shall be returned.

52855 **ERRORS**

52856 The *posix_mem_offset*( ) function shall fail if:

52857 [EACCES] The process has not mapped a memory object supported by this function at
52858 the given address *addr*.

52859 This function shall not return an error code of [EINTR].

52860 **EXAMPLES**

52861 None.

52862 **APPLICATION USAGE**

52863 None.

52864 **RATIONALE**

52865 None.

52866 **FUTURE DIRECTIONS**

52867 None.

52868 **SEE ALSO**

52869 *mmap( )*, *posix_typed_mem_open( )*

52870 XBD **<sys/mman.h>**

52871 **CHANGE HISTORY**

52872 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

NAME

posix_memalign — aligned memory allocation (**ADVANCED REALTIME**)

SYNOPSIS

ADV
```
#include <stdlib.h>

int posix_memalign(void **memptr, size_t alignment, size_t size);
```

DESCRIPTION

The *posix_memalign*( ) function shall allocate *size* bytes aligned on a boundary specified by *alignment*, and shall return a pointer to the allocated memory in *memptr*. The value of *alignment* shall be a power of two multiple of *sizeof* (**void** *).

Upon successful completion, the value pointed to by *memptr* shall be a multiple of *alignment*.

If the size of the space requested is 0, the behavior is implementation-defined: either a null pointer shall be returned in *memptr*, or the behavior shall be as if the size were some non-zero value, except that the behavior is undefined if the the value returned in *memptr* is used to access an object.

CX The *free*( ) function shall deallocate memory that has previously been allocated by *posix_memalign*( ).

For purposes of determining the existence of a data race, *posix_memalign*( ) shall behave as though it accessed only memory locations accessible through its argument and not other static duration storage. The function may, however, visibly modify the storage that it allocates. Calls to *aligned_alloc*( ), *calloc*( ), *free*( ), *malloc*( ), *posix_memalign*( ), *reallocarray*( ), and *realloc*( ) that allocate or deallocate a particular region of memory shall occur in a single total order (see Section 4.15.1, on page 100), and each such deallocation call shall synchronize with the next allocation (if any) in this order.

RETURN VALUE

Upon successful completion, *posix_memalign*( ) shall return zero; otherwise, an error number shall be returned to indicate the error and the contents of *memptr* shall either be left unmodified or be set to a null pointer.

If *size* is 0, either:

- *posix_memalign*( ) shall not attempt to allocate any space, in which case either an implementation-defined error number shall be returned, or zero shall be returned with a null pointer returned in *memptr*, or
- *posix_memalign*( ) shall attempt to allocate some space and, if the allocation succeeds, zero shall be returned and a pointer to the allocated space shall be returned in *memptr*. The application shall ensure that the pointer is not used to access an object.

ERRORS

The *posix_memalign*( ) function shall fail if:

[EINVAL] The value of the alignment parameter is not a power of two multiple of *sizeof* (**void** *).

[ENOMEM] There is insufficient memory available with the requested alignment.

EXAMPLES

The following example shows how applications can obtain consistent behavior on error by setting **memptr* to be a null pointer before calling *posix_memalign*( ).

```
void *ptr = NULL;
...
//do some work, which might goto error
if (posix_memalign(&ptr, align, size))
    goto error;
//do some more work, which might goto error
...
error:
    free(ptr);
    //more cleanup;
```

APPLICATION USAGE

The *posix_memalign*( ) function is part of the Advisory Information option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aligned_alloc( ), *free*( ), *malloc*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

Issue 7

Austin Group Interpretation 1003.1-2001 #058 is applied, clarifying the value of the *alignment* argument in the DESCRIPTION.

Austin Group Interpretation 1003.1-2001 #152 is applied, clarifying the behavior when the size of the space requested is 0.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0251 [526] and XSH/TC2-2008/0252 [520,526] are applied.

Issue 8

Austin Group Defect 1302 is applied, aligning this function with the requirements of the ISO/IEC 9899:2018 standard on other memory allocation functions.

NAME

posix_openpt — open a pseudo-terminal device

SYNOPSIS

XSI
```
#include <stdlib.h>

int posix_openpt(int oflag);
```

DESCRIPTION

The *posix_openpt*( ) function shall establish a connection between a manager device for a pseudo-terminal and a file descriptor. The file descriptor shall be allocated as described in Section 2.6 (on page 525) and can be used by other I/O functions that refer to that pseudo-terminal.

The file status flags and file access modes of the open file description shall be set according to the value of *oflag*.

Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list:

O_RDWR Open for reading and writing.

O_CLOEXEC Atomically set the FD_CLOEXEC flag on the file descriptor.

O_CLOFORK Atomically set the FD_CLOFORK flag on the file descriptor.

O_NOCTTY If set *posix_openpt*( ) shall not cause the terminal device to become the controlling terminal for the process.

The behavior of other values for the *oflag* argument is unspecified.

RETURN VALUE

Upon successful completion, the *posix_openpt*( ) function shall open a file descriptor for a manager pseudo-terminal device and return a non-negative integer representing the file descriptor. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *posix_openpt*( ) function shall fail if:

[EMFILE] All file descriptors available to the process are currently open.

[ENFILE] The maximum allowable number of files is currently open in the system.

The *posix_openpt*( ) function may fail if:

[EINVAL] The value of *oflag* is not valid.

[EAGAIN] Out of pseudo-terminal resources.

EXAMPLES

Opening a Pseudo-Terminal and Returning the Name of the Subsidiary Device and a File Descriptor

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

int managerfd, subsidiaryfd;
char *subsidiarydevice;

managerfd = posix_openpt(O_RDWR|O_NOCTTY);

if (managerfd == -1
    || grantpt (managerfd) == -1
```



```
            || unlockpt (managerfd) == -1
            || (subsidiarydevice = ptsname (managerfd)) == NULL)
            return -1;
        printf("subsidiary device is: %s\n", subsidiarydevice);
        subsidiaryfd = open(subsidiarydevice, O_RDWR|O_NOCTTY);
        if (subsidiaryfd < 0)
            return -1;
```

APPLICATION USAGE

This function is a method for portably obtaining a file descriptor of a manager terminal device for a pseudo-terminal. The *grantpt*( ) function and the *ptsname*( ) and *ptsname_r*( ) functions can be used to manipulate mode and ownership permissions, and to obtain the name of the subsidiary device, respectively.

RATIONALE

The standard developers considered the matter of adding a special device for cloning manager pseudo-terminal devices: the **/dev/ptmx** device. However, consensus could not be reached, and it was felt that adding a new function would permit other implementations. The *posix_openpt*( ) function is designed to complement the *grantpt*( ), *ptsname*( ), *ptsname_r*( ), and *unlockpt*( ) functions.

On implementations supporting the **/dev/ptmx** clone device, opening the manager device of a pseudo-terminal is simply:

```
mfdp = open("/dev/ptmx", oflag );
if (mfdp < 0)
    return -1;
```

The O_CLOEXEC and O_CLOFORK flags are necessary to avoid a data race in multi-threaded applications. Without O_CLOFORK, a file descriptor is leaked into a child process created by one thread in the window between another thread creating a file descriptor with *posix_openpt*( ) and then using *fcntl*( ) to set the FD_CLOFORK flag. Without O_CLOEXEC, a file descriptor intentionally inherited by child processes is similarly leaked into an executed program if FD_CLOEXEC is not set atomically.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.6 (on page 525), *grantpt*( ), *open*( ), *ptsname*( ), *unlockpt*( )

XBD **<fcntl.h>**, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 6.

Issue 7

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-51 is applied, correcting an error in the EXAMPLES section.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0253 [835] and XSH/TC2-2008/0254 [835] are applied.

Issue 8

Austin Group Defects 411 and 1318 are applied, adding O_CLOEXEC and O_CLOFORK.

Austin Group Defect 508 is applied, changing the APPLICATION USAGE and RATIONALE sections to refer to *ptsname_r*( ) as well as *ptsname*( ).

Austin Group Defect 593 is applied, removing `#include <fcntl.h>` from the SYNOPSIS and a reference to **<fcntl.h>** from the DESCRIPTION section.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1466 is applied, changing the terminology used for pseudo-terminal devices.

NAME

posix_spawn, posix_spawnp — spawn a process (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <spawn.h>

int posix_spawn(pid_t *restrict pid, const char *restrict path,
    const posix_spawn_file_actions_t *restrict file_actions,
    const posix_spawnattr_t *restrict attrp,
    char *const argv[restrict], char *const envp[restrict]);
int posix_spawnp(pid_t *restrict pid, const char *restrict file,
    const posix_spawn_file_actions_t *restrict file_actions,
    const posix_spawnattr_t *restrict attrp,
    char *const argv[restrict], char *const envp[restrict]);
```

DESCRIPTION

The *posix_spawn*( ) and *posix_spawnp*( ) functions shall create a new process (child process) from the specified process image. The new process image shall be constructed from a regular executable file called the new process image file.

When a C program is executed as the result of this call, it shall be entered as a C-language function call as follows:

```
int main(int argc, char *argv[]);
```

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. In addition, the following variable:

```
extern char **environ;
```

shall be initialized as a pointer to an array of character pointers to the environment strings.

The argument *argv* is an array of character pointers to null-terminated strings. The last member of this array shall be a null pointer and is not counted in *argc*. These strings constitute the argument list available to the new process image. The value in *argv*[0] should point to a filename string that is associated with the process image being started by the *posix_spawn*( ) or *posix_spawnp*( ) function.

The argument *envp* is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process image. The environment array is terminated by a null pointer.

The number of bytes available for the combined argument and environment lists of the child process is {ARG_MAX}. The implementation shall specify in the system documentation (see XBD Chapter 2, on page 15) whether any list overhead, such as length words, null terminators, pointers, or alignment bytes, is included in this total.

The *path* argument to *posix_spawn*( ) is a pathname that identifies the new process image file to execute; if the pathname does not start with a <slash>, it shall be interpreted relative to the working directory of the child process after all *file_actions* have been performed.

The *file* parameter to *posix_spawnp*( ) shall be used to construct a pathname that identifies the new process image file. If the *file* parameter contains a <slash> character, the *file* parameter shall be used as the pathname for the new process image file. Otherwise, the path prefix for this file shall be obtained by a search of the directories passed as the environment variable *PATH* (see XBD Chapter 8, on page 167), using the working directory of the child process after all *file_actions* have been performed. If this environment variable is not defined, the results of the search are implementation-defined. However, if at least one of the *exec* family of functions

would fail with [ENOEXEC] because the process image contents are not executable, this shall cause *posix_spawnp*( ) to fail rather than attempting a fallback to invoking the process as a shell script passed to *sh*.

If *file_actions* is a null pointer, then file descriptors open in the calling process shall remain open in the child process, except for those whose FD_CLOEXEC or FD_CLOFORK flag is set (see *fcntl*( )), and except for file descriptors that are closed by a fork handler (if fork handlers are called). For those file descriptors that remain open, the child process shall not inherit any process-owned file locks, but all remaining attributes of the corresponding open file descriptions (see *fcntl*( )), shall remain unchanged. The current working directory of the child process shall be the same as it is in the parent process.

If *file_actions* is not a null pointer, then the file descriptors open in the child process shall be those open in the calling process as modified by FD_CLOFORK file descriptor flags, fork handlers (if they are called), the spawn file actions object pointed to by *file_actions*, and the FD_CLOEXEC flag of each remaining open file descriptor after the spawn file actions have been processed. The effective order of processing the spawn file actions shall be:

1. The set of open file descriptors for the child process shall initially be the same set as is open for the calling process, except for those that have the FD_CLOFORK flag set and any that are closed by fork handlers (if they are called). The child process shall not inherit any file locks, but all remaining attributes of the corresponding open file descriptions (see *fcntl*( )), shall remain unchanged.
2. The signal mask, signal default actions, and the effective user and group IDs for the child process shall be changed as specified in the attributes object referenced by *attrp*.
3. The file actions specified by the spawn file actions object shall be performed in the order in which they were added to the spawn file actions object, and relative pathnames in a given action shall be interpreted in relation to the tracked working directory. The tracked working directory shall begin with the current working directory of the parent process, and can be altered according to *chdir* or *fchdir* file actions; the current working directory of the child process shall be the final state of the tracked working directory after all file actions have been applied.
4. Any file descriptor that has its FD_CLOEXEC flag set shall be closed.

If file descriptor 0, 1, or 2 would otherwise be closed in the new process image created by *posix_spawn*( ) or *posix_spawnp*( ), implementations may open an unspecified file for the file descriptor in the new process image. If a standard utility or a conforming application is executed with file descriptor 0 not open for reading or with file descriptor 1 or 2 not open for writing, the environment in which the utility or application is executed shall be deemed non-conforming, and consequently the utility or application might not behave as described in this standard.

The **posix_spawnattr_t** spawn attributes object type is defined in **<spawn.h>**. It shall contain at least the attributes defined below.

If the POSIX_SPAWN_SETPGROUP flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is non-zero, then the child's process group shall be as specified in the *spawn-pgroup* attribute of the object referenced by *attrp*.

As a special case, if the POSIX_SPAWN_SETPGROUP flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is set to zero, then the child shall be in a new process group with a process group ID equal to its process ID.

If the POSIX_SPAWN_SETSID flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, the child process shall be the session leader of a new session, shall be the process group

leader of a new process group, and shall have no controlling terminal. The process group ID of the child process shall be set equal to the process ID of the child process. The child process shall be the only process in the new process group and the only process in the new session.

If both the POSIX_SPAWN_SETPGROUP flag and the POSIX_SPAWN_SETSID flag are set in the *spawn-flags* attribute of the object referenced by *attrp*, the behavior is unspecified.

If neither the POSIX_SPAWN_SETPGROUP flag nor the POSIX_SPAWN_SETSID flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, the new child process shall inherit the parent's process group.

PS If the POSIX_SPAWN_SETSCHEDPARAM flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, but POSIX_SPAWN_SETSCHEDULER is not set, the new process image shall initially have the scheduling policy of the calling process with the scheduling parameters specified in the *spawn-schedparam* attribute of the object referenced by *attrp*.

If the POSIX_SPAWN_SETSCHEDULER flag is set in the *spawn-flags* attribute of the object referenced by *attrp* (regardless of the setting of the POSIX_SPAWN_SETSCHEDPARAM flag), the new process image shall initially have the scheduling policy specified in the *spawn-schedpolicy* attribute of the object referenced by *attrp* and the scheduling parameters specified in the *spawn-schedparam* attribute of the same object.

The POSIX_SPAWN_RESETIDS flag in the *spawn-flags* attribute of the object referenced by *attrp* governs the effective user ID of the child process. If this flag is not set, the child process shall inherit the effective user ID of the parent process. If this flag is set, the effective user ID of the child process shall be reset to the parent's real user ID. In either case, if the set-user-ID mode bit of the new process image file is set, the effective user ID of the child process shall become that file's owner ID before the new process image begins execution.

The POSIX_SPAWN_RESETIDS flag in the *spawn-flags* attribute of the object referenced by *attrp* also governs the effective group ID of the child process. If this flag is not set, the child process shall inherit the effective group ID of the parent process. If this flag is set, the effective group ID of the child process shall be reset to the parent's real group ID. In either case, if the set-group-ID mode bit of the new process image file is set, the effective group ID of the child process shall become that file's group ID before the new process image begins execution.

If the POSIX_SPAWN_SETSIGMASK flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, the child process shall initially have the signal mask specified in the *spawn-sigmask* attribute of the object referenced by *attrp*.

If the POSIX_SPAWN_SETSIGDEF flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, the signals specified in the *spawn-sigdefault* attribute of the same object shall be set to their default actions in the child process. Signals set to the default action in the parent process shall be set to the default action in the child process.

Signals set to be caught by the calling process shall be set to the default action in the child process.

Except for SIGCHLD, signals set to be ignored by the calling process image shall be set to be ignored by the child process, unless otherwise specified by the POSIX_SPAWN_SETSIGDEF flag being set in the *spawn-flags* attribute of the object referenced by *attrp* and the signals being indicated in the *spawn-sigdefault* attribute of the object referenced by *attrp*.

If the SIGCHLD signal is set to be ignored by the calling process, it is unspecified whether the SIGCHLD signal is set to be ignored or to the default action in the child process, unless otherwise specified by the POSIX_SPAWN_SETSIGDEF flag being set in the *spawn_flags* attribute of the object referenced by *attrp* and the SIGCHLD signal being indicated in the

53175 *spawn_sigdefault* attribute of the object referenced by *attrp*.

53176 If the value of the *attrp* pointer is a null pointer, then the default values are used.

53177 All process attributes, other than those influenced by the attributes set in the object referenced
53178 by *attrp* as specified above or by the file descriptor manipulations specified in *file_actions*, shall
53179 appear in the new process image as though *fork*( ) had been called to create a child process and
53180 then a member of the *exec* family of functions had been called by the child process to execute the
53181 new process image.

53182 It is implementation-defined whether the fork handlers are run when *posix_spawn*( ) or
53183 *posix_spawnp*( ) is called.

53184 **RETURN VALUE**

53185 Upon successful completion, *posix_spawn*( ) and *posix_spawnp*( ) shall return the process ID of the
53186 child process to the parent process, in the variable pointed to by a non-null *pid* argument, and
53187 shall return zero as the function return value. Otherwise, no child process shall be created, the
53188 value stored into the variable pointed to by a non-null *pid* is unspecified, and an error number
53189 shall be returned as the function return value to indicate the error. If the *pid* argument is a null
53190 pointer, the process ID of the child is not returned to the caller.

53191 **ERRORS**

53192 These functions may fail if:

53193 [EINVAL] The value specified by *file_actions* or *attrp* is invalid.

53194 If this error occurs after the calling process successfully returns from the *posix_spawn*( ) or
53195 *posix_spawnp*( ) function, the child process may exit with exit status 127.

53196 If *posix_spawn*( ) or *posix_spawnp*( ) fail for any of the reasons that would cause *fork*( ) or one of
53197 the *exec* family of functions to fail, including when the corresponding *exec* function would
53198 attempt a fallback to *sh* instead of failing with [ENOEXEC], an error value shall be returned as
53199 described by *fork*( ) and *exec*, respectively; or, if the error occurs after the calling process
53200 successfully returns, the child process shall exit with exit status 127.

53201 If POSIX_SPAWN_SETPGROUP is set in the *spawn-flags* attribute of the object referenced by
53202 *attrp*, and *posix_spawn*( ) or *posix_spawnp*( ) fails while changing the child's process group, an
53203 error value shall be returned as described by *setpgid*( ); or, if the error occurs after the calling
53204 process successfully returns, the child process shall exit with exit status 127.

53205 If POSIX_SPAWN_SETSID is set in the *spawn-flags* attribute of the object referenced by *attrp*, and
53206 *posix_spawn*( ) or *posix_spawnp*( ) fails while creating the new session, changing the child's
53207 session ID, or changing the child's process group, an error value shall be returned as described
53208 by *setsid*( ); or, if the error occurs after the calling process successfully returns, the child process
53209 shall exit with exit status 127.

53210 PS If POSIX_SPAWN_SETSCHEDPARAM is set and POSIX_SPAWN_SETSCHEDULER is not set in
53211 the *spawn-flags* attribute of the object referenced by *attrp*, then if *posix_spawn*( ) or *posix_spawnp*( )
53212 fails for any of the reasons that would cause *sched_setparam*( ) to fail, an error value shall be
53213 returned as described by *sched_setparam*( ); or, if the error occurs after the calling process
53214 successfully returns, the child process shall exit with exit status 127.

53215 If POSIX_SPAWN_SETSCHEDULER is set in the *spawn-flags* attribute of the object referenced by
53216 *attrp*, and if *posix_spawn*( ) or *posix_spawnp*( ) fails for any of the reasons that would cause
53217 *sched_setscheduler*( ) to fail, an error value shall be returned as described by *sched_setscheduler*( );
53218 or, if the error occurs after the calling process successfully returns, the child process shall exit
53219 with exit status 127.

If the *file_actions* argument is not a null pointer, and specifies any *chdir*, *close*, *dup2*, *fchdir*, or *open* actions to be performed, and if *posix_spawn*( ) or *posix_spawnp*( ) fails for any of the reasons that would cause *chdir*( ), *close*( ), *dup2*( ), *fchdir*( ), or *open*( ) to fail, other than attempting a *close*( ) on a file descriptor that is in range but already closed, an error value shall be returned as described by *chdir*( ), *close*( ), *dup2*( ), *fchdir*( ), and *open*( ), respectively; or, if the error occurs after the calling process successfully returns, the child process shall exit with exit status 127. An open file action may, by itself, result in any of the errors described by *close*( ) or *dup2*( ), in addition to those described by *open*( ).

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

See also the APPLICATION USAGE section for *exec*.

RATIONALE

The *posix_spawn*( ) function and its close relation *posix_spawnp*( ) have been introduced to overcome the following perceived difficulties with *fork*( ): the *fork*( ) function is difficult or impossible to implement without swapping or dynamic address translation.

- Swapping is generally too slow for a realtime environment.
- Dynamic address translation is not available everywhere that POSIX might be useful.
- Processes are too useful to simply option out of POSIX whenever it must run without address translation or other MMU services.

Thus, POSIX needs process creation and file execution primitives that can be efficiently implemented without address translation or other MMU services.

The *posix_spawn*( ) function is implementable as a library routine, but both *posix_spawn*( ) and *posix_spawnp*( ) are designed as kernel operations. Also, although they may be an efficient replacement for many *fork*( )/*exec* pairs, their goal is to provide useful process creation primitives for systems that have difficulty with *fork*( ), not to provide drop-in replacements for *fork*( )/*exec*.

This view of the role of *posix_spawn*( ) and *posix_spawnp*( ) influenced the design of their API. It does not attempt to provide the full functionality of *fork*( )/*exec* in which arbitrary user-specified operations of any sort are permitted between the creation of the child process and the execution of the new process image; any attempt to reach that level would need to provide a programming language as parameters. Instead, *posix_spawn*( ) and *posix_spawnp*( ) are process creation primitives like the *Start_Process* and *Start_Process_Search* Ada language bindings package *POSIX_Process_Primitives* and also like those in many operating systems that are not UNIX systems, but with some POSIX-specific additions.

To achieve its coverage goals, *posix_spawn*( ) and *posix_spawnp*( ) have control of seven types of inheritance: file descriptors, current working directory, process group ID, user and group ID, signal mask, scheduling, and whether each signal ignored in the parent will remain ignored in the child, or be reset to its default action in the child.

Control of file descriptors is required to allow an independently written child process image to access data streams opened by and even generated or read by the parent process without being specifically coded to know which parent files and file descriptors are to be used. Control of the current working directory is required because the parent process may want to constrain the resources that the child process can reach from its current working directory or affect how relative pathnames are interpreted, while recognizing that a multi-threaded parent process

would require a lot of overhead to safely change its own working directory prior to creating the child process. Control of the process group ID is required to control how the job control of the child process relates to that of the parent.

Control of the signal mask and signal defaulting is sufficient to support the implementation of *system*( ). Although support for *system*( ) is not explicitly one of the goals for *posix_spawn*( ) and *posix_spawnp*( ), it is covered under the ``at least 50%'' coverage goal.

The intention is that the normal file descriptor inheritance across *fork*( ), the subsequent effect of the specified spawn file actions, and the normal file descriptor inheritance across one of the *exec* family of functions should fully specify open file inheritance. The implementation need make no decisions regarding the set of open file descriptors when the child process image begins execution, those decisions having already been made by the caller and expressed as the set of open file descriptors and their FD_CLOEXEC and FD_CLOFORK flags at the time of the call, the actions of fork handlers (if they are called), and the spawn file actions object specified in the call. We have been assured that in cases where the POSIX *Start_Process* Ada primitives have been implemented in a library, this method of controlling file descriptor inheritance may be implemented very easily.

We can identify several problems with *posix_spawn*( ) and *posix_spawnp*( ), but there does not appear to be a solution that introduces fewer problems. Environment modification for child process attributes not specifiable via the *attrp* or *file_actions* arguments must be done in the parent process, and since the parent generally wants to save its context, it is more costly than similar functionality with *fork*( )/*exec*. It is also complicated to modify the environment of a multi-threaded process temporarily, since all threads must agree when it is safe for the environment to be changed. However, this cost is only borne by those invocations of *posix_spawn*( ) and *posix_spawnp*( ) that use the additional functionality. Since extensive modifications are not the usual case, and are particularly unlikely in time-critical code, keeping much of the environment control out of *posix_spawn*( ) and *posix_spawnp*( ) is appropriate design.

The *posix_spawn*( ) and *posix_spawnp*( ) functions do not have all the power of *fork*( )/*exec*. This is to be expected. The *fork*( ) function is a wonderfully powerful operation. We do not expect to duplicate its functionality in a simple, fast function with no special hardware requirements. It is worth noting that *posix_spawn*( ) and *posix_spawnp*( ) are very similar to the process creation operations on many operating systems that are not UNIX systems.

Requirements

The requirements for *posix_spawn*( ) and *posix_spawnp*( ) are:

- They must be implementable without an MMU or unusual hardware.
- They must be compatible with existing POSIX standards.

Additional goals are:

- They should be efficiently implementable.
- They should be able to replace at least 50% of typical executions of *fork*( ).
- A system with *posix_spawn*( ) and *posix_spawnp*( ) and without *fork*( ) should be useful, at least for realtime applications.
- A system with *fork*( ) and the *exec* family should be able to implement *posix_spawn*( ) and *posix_spawnp*( ) as library routines.

Two-Syntax

POSIX *exec* has several calling sequences with approximately the same functionality. These appear to be required for compatibility with existing practice. Since the existing practice for the *posix_spawn**( ) functions is otherwise substantially unlike POSIX, we feel that simplicity outweighs compatibility. There are, therefore, only two names for the *posix_spawn**( ) functions.

The parameter list does not differ between *posix_spawn*( ) and *posix_spawnp*( ); *posix_spawnp*( ) interprets the second parameter more elaborately than *posix_spawn*( ).

Compatibility with POSIX.5 (Ada)

The *Start_Process* and *Start_Process_Search* procedures from the *POSIX_Process_Primitives* package from the Ada language binding to POSIX.1 encapsulate *fork*( ) and *exec* functionality in a manner similar to that of *posix_spawn*( ) and *posix_spawnp*( ). Originally, in keeping with our simplicity goal, the standard developers had limited the capabilities of *posix_spawn*( ) and *posix_spawnp*( ) to a subset of the capabilities of *Start_Process* and *Start_Process_Search*; certain non-default capabilities were not supported. However, based on suggestions by the ballot group to improve file descriptor mapping or drop it, and on the advice of an Ada Language Bindings working group member, the standard developers decided that *posix_spawn*( ) and *posix_spawnp*( ) should be sufficiently powerful to implement *Start_Process* and *Start_Process_Search*. The rationale is that if the Ada language binding to such a primitive had already been approved as an IEEE standard, there can be little justification for not approving the functionally-equivalent parts of a C binding. The only three capabilities provided by *posix_spawn*( ) and *posix_spawnp*( ) that are not provided by *Start_Process* and *Start_Process_Search* are optionally specifying the child's process group ID, the set of signals to be reset to default signal handling in the child process, and the child's scheduling policy and parameters.

For the Ada language binding for *Start_Process* to be implemented with *posix_spawn*( ), that binding would need to explicitly pass an empty signal mask and the parent's environment to *posix_spawn*( ) whenever the caller of *Start_Process* allowed these arguments to default, since *posix_spawn*( ) does not provide such defaults. The ability of *Start_Process* to mask user-specified signals during its execution is functionally unique to the Ada language binding and must be dealt with in the binding separately from the call to *posix_spawn*( ).

Process Group

The process group inheritance field can be used to join the child process with an existing process group. By assigning a value of zero to the *spawn-pgroup* attribute of the object referenced by *attrp*, the *setpgid*( ) mechanism will place the child process in a new process group.

Threads

Without the *posix_spawn*( ) and *posix_spawnp*( ) functions, systems without address translation can still use threads to give an abstraction of concurrency. In many cases, thread creation suffices, but it is not always a good substitute. The *posix_spawn*( ) and *posix_spawnp*( ) functions are considerably ``heavier'' than thread creation. Processes have several important attributes that threads do not. Even without address translation, a process may have base-and-bound memory protection. Each process has a process environment including security attributes and file capabilities, and powerful scheduling attributes. Processes abstract the behavior of non-uniform-memory-architecture multi-processors better than threads, and they are more convenient to use for activities that are not closely linked.

The *posix_spawn*( ) and *posix_spawnp*( ) functions may not bring support for multiple processes to every configuration. Process creation is not the only piece of operating system support required

to support multiple processes. The total cost of support for multiple processes may be quite high in some circumstances. Existing practice shows that support for multiple processes is uncommon and threads are common among ``tiny kernels''. There should, therefore, probably continue to be AEPs for operating systems with only one process.

Asynchronous Error Notification

A library implementation of *posix_spawn*( ) or *posix_spawnp*( ) may not be able to detect all possible errors before it forks the child process. POSIX.1-2024 provides for an error indication returned from a child process which could not successfully complete the spawn operation via a special exit status which may be detected using the status value returned by *wait*( ), *waitid*( ), and *waitpid*( ).

The *stat_val* interface and the macros used to interpret it are not well suited to the purpose of returning API errors, but they are the only path available to a library implementation. Thus, an implementation may cause the child process to exit with exit status 127 for any error detected during the spawn process after the *posix_spawn*( ) or *posix_spawnp*( ) function has successfully returned.

The standard developers had proposed using two additional macros to interpret *stat_val*. The first, WIFSPAWNFAIL, would have detected a status that indicated that the child exited because of an error detected during the *posix_spawn*( ) or *posix_spawnp*( ) operations rather than during actual execution of the child process image; the second, WSPAWNERRNO, would have extracted the error value if WIFSPAWNFAIL indicated a failure. Unfortunately, the ballot group strongly opposed this because it would make a library implementation of *posix_spawn*( ) or *posix_spawnp*( ) dependent on kernel modifications to *waitpid*( ) to be able to embed special information in *stat_val* to indicate a spawn failure.

The 8 bits of child process exit status that are guaranteed by POSIX.1-2024 to be accessible to the waiting parent process are insufficient to disambiguate a spawn error from any other kind of error that may be returned by an arbitrary process image. No other bits of the exit status are required to be visible in *stat_val*, so these macros could not be strictly implemented at the library level. Reserving an exit status of 127 for such spawn errors is consistent with the use of this value by *system*( ) and *popen*( ) to signal failures in these operations that occur after the function has returned but before a shell is able to execute. The exit status of 127 does not uniquely identify this class of error, nor does it provide any detailed information on the nature of the failure. Note that a kernel implementation of *posix_spawn*( ) or *posix_spawnp*( ) is permitted (and encouraged) to return any possible error as the function value, thus providing more detailed failure information to the parent process.

Thus, no special macros are available to isolate asynchronous *posix_spawn*( ) or *posix_spawnp*( ) errors. Instead, errors detected by the *posix_spawn*( ) or *posix_spawnp*( ) operations in the context of the child process before the new process image executes are reported by setting the child's exit status to 127. The calling process may use the WIFEXITED and WEXITSTATUS macros on the *stat_val* stored by the *wait*( ) or *waitpid*( ) functions to detect spawn failures to the extent that other status values with which the child process image may exit (before the parent can conclusively determine that the child process image has begun execution) are distinct from exit status 127.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *chmod*( ), *close*( ), *dup*( ), *exec*, *exit*( ), *fcntl*( ), *fork*( ), *fstatat*( ), *kill*( ), *open*( ), *posix_spawn_file_actions_addchdir*( ), *posix_spawn_file_actions_addclose*( ), *posix_spawn_file_actions_adddup2*( ), *posix_spawn_file_actions_destroy*( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ), *posix_spawnattr_getsigmask*( ), *sched_setparam*( ), *sched_setscheduler*( ), *setpgid*( ), *setsid*( ), *setuid*( ), *times*( ), *wait*( ), *waitid*( )

XBD Chapter 8 (on page 167), **<spawn.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #103 is applied, noting that the signal default actions are changed as well as the signal mask in step 2.

IEEE PASC Interpretation 1003.1 #132 is applied.

Issue 7

Functionality relating to the Threads option is moved to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0433 [291], XSH/TC1-2008/0434 [173], and XSH/TC1-2008/0435 [173] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0255 [824] is applied.

Issue 8

Austin Group Defect 370 is applied, requiring that attempting to close a file descriptor that is in range but already closed is not treated as an error.

Austin Group Defect 768 is applied, adding OFD-owned file locks.

Austin Group Defect 1044 is applied, adding POSIX_SPAWN_SETSID.

Austin Group Defect 1208 is applied, adding *posix_spawn_file_actions_addchdir*( ) and *posix_spawn_file_actions_addfchdir*( ).

Austin Group Defect 1318 is applied, adding FD_CLOFORK and clarifying that the inherited set of file descriptors is affected by the actions of fork handlers (if they are called).

Austin Group Defect 1328 is applied, adding the **restrict** keyword to the third parameter of *posix_spawn*( ) and *posix_spawnp*( ).

Austin Group Defect 1362 is applied, removing parentheses around some text intended to be normative.

Austin Group Defect 1674 is applied, adding a requirement that *posix_spawnp*( ) does not fallback to executing *sh* when the corresponding *exec* function would do so instead of failing with [ENOEXEC].

NAME

posix_spawn_file_actions_addchdir, posix_spawn_file_actions_addfchdir — add chdir or fchdir action to spawn file actions object (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <spawn.h>

int posix_spawn_file_actions_addchdir(posix_spawn_file_actions_t
    *restrict file_actions, const char *restrict path);
int posix_spawn_file_actions_addfchdir(posix_spawn_file_actions_t
    *file_actions, int fildes);
```

DESCRIPTION

The *posix_spawn_file_actions_addchdir*( ) function shall add a *chdir* action to the object referenced by *file_actions* that shall cause the working directory to be set to *path* (as if *chdir*(*path*) had been called) when a new process is spawned using this file actions object. A relative *path* shall be interpreted in relation to the working directory determined by any prior actions. The string pointed to by *path* shall be copied by the *posix_spawn_file_actions_addchdir*( ) function.

The *posix_spawn_file_actions_addfchdir*( ) function shall add an *fchdir* action to the object referenced by *file_actions* that shall cause the working directory to be set to *fildes* (as if *fchdir*(*fildes*) had been called) when a new process is spawned using this file actions object.

A spawn file actions object is as defined in *posix_spawn_file_actions_addclose*( ).

RETURN VALUE

Upon successful completion, these functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_spawn_file_actions_addfchdir*( ) function shall fail if:

[EBADF] The value specified by *fildes* is negative.

These functions shall fail if:

[ENOMEM] Insufficient memory exists to add to the spawn file actions object.

These functions may fail if:

[EINVAL] The value specified by *file_actions* is invalid.

It shall not be considered an error for the *path* or *fildes* argument passed to these functions to specify a pathname or file descriptor for which the specified operation could not be performed at the time of the call. Any such error shall be detected when the associated file actions object is later used during a *posix_spawn*( ) or *posix_spawnp*( ) operation.

EXAMPLES

None.

APPLICATION USAGE

The *posix_spawn_file_actions_addchdir*( ) and *posix_spawn_file_actions_addfchdir*( ) functions are part of the Spawn option and need not be provided on all implementations.

Changing the working directory of a child process can be useful when invoking utilities such as *pax*. Furthermore, the ability to add *fchdir* actions to *posix_spawn*( ) gives the caller as much control over relative pathnames processed in the context of the child as it would otherwise have using *openat*( ), since all file actions are processed in sequence in the context of the child at a point where the child process is still single-threaded. Without *chdir* or *fchdir* actions, changing the working directory of the child would require a shim utility (some implementations provide


```
env -C /new/path program args...
```

as an extension, but the standard does not require this extension), or else temporarily changing the working directory in the parent process prior to calling *posix_spawn*( ) (but this requires locking in a multi-threaded process, to ensure that no other thread is impacted by the temporary change to global state).

File actions are performed in a new process created by *posix_spawn*( ) or *posix_spawnp*( ) in the same order that they were added to the file actions object. Thus, the execution of an *open* action that was created by a call to *posix_spawn_file_actions_addopen*( ) that specifies a relative path will be affected by the execution of a *chdir* or *fchdir* action that was created by a previous call to *posix_spawn_file_actions_addchdir*( ) or *posix_spawn_file_actions_addfchdir*( ). Likewise, a relative path passed to *posix_spawn*( ) will be affected by the last *chdir* or *fchdir* action in the file action list.

RATIONALE

Refer to the RATIONALE section in *posix_spawn_file_actions_addclose*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

chdir( ), *fchdir*( ), *posix_spawn*( ), *posix_spawn_file_actions_addclose*( ), *posix_spawn_file_actions_destroy*( )

XBD **<spawn.h>**

CHANGE HISTORY

First released in Issue 8. Derived from Solaris *posix_spawn_file_actions_addchdir_np*.

NAME

posix_spawn_file_actions_addclose, posix_spawn_file_actions_addopen — add close or open action to spawn file actions object (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <spawn.h>

int posix_spawn_file_actions_addclose(posix_spawn_file_actions_t
    *file_actions, int fildes);
int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t
    *restrict file_actions, int fildes,
    const char *restrict path, int oflag, mode_t mode);
```

DESCRIPTION

These functions shall add a *close* or *open* action to a spawn file actions object.

A spawn file actions object is of type **posix_spawn_file_actions_t** (defined in **<spawn.h>**) and is used to specify a series of actions to be performed by a *posix_spawn*( ) or *posix_spawnp*( ) operation in order to arrive at the set of open file descriptors for the child process given the set of open file descriptors of the parent. POSIX.1-2024 does not define comparison or assignment operators for the type **posix_spawn_file_actions_t**.

A spawn file actions object, when passed to *posix_spawn*( ) or *posix_spawnp*( ), shall specify how the set of open file descriptors in the calling process is transformed into a set of potentially open file descriptors for the spawned process. This transformation shall be as if the specified sequence of actions was performed exactly once, in the context of the spawned process (prior to execution of the new process image), in the order in which the actions were added to the object; additionally, when the new process image is executed, any file descriptor (from this new set) which has its FD_CLOEXEC flag set shall be closed (see *posix_spawn*( )).

The *posix_spawn_file_actions_addclose*( ) function shall add a *close* action to the object referenced by *file_actions* that shall cause the file descriptor *fildes* to be closed (as if *close*(*fildes*) had been called) when a new process is spawned using this file actions object, except that a non-negative *fildes* less than {OPEN_MAX} that is already closed at the time when the new process is spawned shall be ignored rather than failing with [EBADF].

The *posix_spawn_file_actions_addopen*( ) function shall add an *open* action to the object referenced by *file_actions* that shall cause the file named by *path* to be opened (as if *open*(*path*, *oflag*, *mode*) had been called, and the returned file descriptor, if not *fildes*, had been changed to *fildes*) when a new process is spawned using this file actions object. If *fildes* was already an open file descriptor, it shall be closed before the new file is opened. A relative *path* shall be interpreted in relation to the working directory determined by any prior actions.

The string pointed to by *path* shall be copied by the *posix_spawn_file_actions_addopen*( ) function.

RETURN VALUE

Upon successful completion, these functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_spawn_file_actions_addopen*( ) function shall fail if:

[EBADF] The value specified by *fildes* is negative or greater than or equal to {OPEN_MAX}.

The *posix_spawn_file_actions_addclose*( ) function shall fail if:

[EBADF] The value specified by *fildes* is negative.

These functions shall fail if:

[ENOMEM] Insufficient memory exists to add to the spawn file actions object.

These functions may fail if:

[EINVAL] The value specified by *file_actions* is invalid.

It shall not be considered an error for the *fildes* argument passed to these functions to specify a file descriptor for which the specified operation could not be performed at the time of the call. Any such error shall be detected when the associated file actions object is later used during a *posix_spawn*( ) or *posix_spawnp*( ) operation.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

Implementations may use file descriptors that must be inherited into child processes for the child process to remain conforming, such as for message catalog or tracing purposes. Therefore, an application that calls *posix_spawn_file_actions_addclose*( ) with an arbitrary integer risks non-conforming behavior, and this function can only portably be used to close file descriptor values that the application has obtained through explicit actions, or for the three file descriptors corresponding to the standard file streams. In order to avoid a race condition of leaking an unintended file descriptor into a child process or executed program, an application should consider opening all file descriptors with the FD_CLOFORK or FD_CLOEXEC flag, or both flags, set unless the file descriptor is intended to be inherited by child processes or executed programs, respectively.

RATIONALE

A spawn file actions object may be initialized to contain an ordered sequence of *chdir*( ), *close*( ), *dup2*( ), *fchdir*( ), and *open*( ) operations to be used by *posix_spawn*( ) or *posix_spawnp*( ) to arrive at the set of open file descriptors and current working directory inherited by the spawned process from the set of open file descriptors and current working directory in the parent at the time of the *posix_spawn*( ) or *posix_spawnp*( ) call. It had been suggested that the *close*( ) and *dup2*( ) operations alone are sufficient to rearrange file descriptors, and that files which need to be opened for use by the spawned process can be handled either by having the calling process open them before the *posix_spawn*( ) or *posix_spawnp*( ) call (and close them after), or by passing pathnames to the spawned process (in *argv*) so that it may open them itself. The standard developers recommend that applications use one of these two methods when practical, since detailed error status on a failed open operation is always available to the application this way. However, the standard developers feel that allowing a spawn file actions object to specify open operations is still appropriate because:

1. It is consistent with equivalent POSIX.5 (Ada) functionality.
2. It supports the I/O redirection paradigm commonly employed by POSIX programs designed to be invoked from a shell. When such a program is the child process, it may not be designed to open files on its own.
3. It allows file opens that might otherwise fail or violate file ownership/access rights if executed by the parent process.

Regarding 2. above, note that the spawn open file action provides to *posix_spawn*( ) and *posix_spawnp*( ) the same capability that the shell redirection operators provide to *system*( ), only

without the intervening execution of a shell; for example:

```
system ("myprog <file1 3<file2");
```

Regarding 3. above, note that if the calling process needs to open one or more files for access by the spawned process, but has insufficient spare file descriptors, then the *open* action is necessary to allow the *open*( ) to occur in the context of the child process after other file descriptors have been closed (that must remain open in the parent).

Additionally, if a parent is executed from a file having a ``set-user-id'' mode bit set and the POSIX_SPAWN_RESETIDS flag is set in the spawn attributes, a file created within the parent process will (possibly incorrectly) have the parent's effective user ID as its owner, whereas a file created via an *open*( ) action during *posix_spawn*( ) or *posix_spawnp*( ) will have the parent's real ID as its owner; and an open by the parent process may successfully open a file to which the real user should not have access or fail to open a file to which the real user should have access.

File Descriptor Mapping

The standard developers had originally proposed using an array which specified the mapping of child file descriptors back to those of the parent. It was pointed out by the ballot group that it is not possible to reshuffle file descriptors arbitrarily in a library implementation of *posix_spawn*( ) or *posix_spawnp*( ) without provision for one or more spare file descriptor entries (which simply may not be available). Such an array requires that an implementation develop a complex strategy to achieve the desired mapping without inadvertently closing the wrong file descriptor at the wrong time.

It was noted by a member of the Ada Language Bindings working group that the approved Ada Language *Start_Process* family of POSIX process primitives use a caller-specified set of file actions to alter the normal *fork*( )/*exec* semantics for inheritance of file descriptors in a very flexible way, yet no such problems exist because the burden of determining how to achieve the final file descriptor mapping is completely on the application. Furthermore, although the file actions interface appears frightening at first glance, it is actually quite simple to implement in either a library or the kernel.

The *posix_spawn_file_actions_addclose*( ) function is not required to check whether the file descriptor is less than {OPEN_MAX} because on some implementations {OPEN_MAX} reflects the RLIMIT_NOFILE soft limit and therefore calling *setrlimit*( ) to reduce this limit can result in an {OPEN_MAX} value less than or equal to an already open file descriptor. Applications need to be able to close such file descriptors on spawn. On implementations where {OPEN_MAX} does not change, it is recommended that *posix_spawn_file_actions_addclose*( ) should return [EBADF] if *fildes* is greater than or equal to {OPEN_MAX}.

FUTURE DIRECTIONS

None.

SEE ALSO

close( ), *dup*( ), *open*( ), *posix_spawn*( ), *posix_spawn_file_actions_addchdir*( ), *posix_spawn_file_actions_adddup2*( ), *posix_spawn_file_actions_destroy*( )

XBD **<spawn.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #105 is applied, adding a note to the DESCRIPTION that the string pointed to by *path* is copied by the *posix_spawn_file_actions_addopen*( ) function.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0436 [418], XSH/TC1-2008/0437 [149], XSH/TC1-2008/0438 [291], and XSH/TC1-2008/0439 [418] are applied.

Issue 8

Austin Group Defect 370 is applied, requiring that attempting to close a file descriptor that is in range, but already closed at the time when the new process is spawned, is not treated as an error.

Austin Group Defect 1208 is applied, adding *posix_spawn_file_actions_addchdir*( ) and *posix_spawn_file_actions_addfchdir*( ).

Austin Group Defect 1318 is applied, adding FD_CLOFORK.

NAME

posix_spawn_file_actions_adddup2 — add dup2 action to spawn file actions object (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <spawn.h>

int posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t
    *file_actions, int fildes, int newfildes);
```

DESCRIPTION

The *posix_spawn_file_actions_adddup2*( ) function shall add a *dup2*( ) action to the object referenced by *file_actions* that shall cause the file descriptor *fildes* to be duplicated as *newfildes* (as if *dup2*(*fildes*, *newfildes*) had been called) when a new process is spawned using this file actions object.

If *fildes* and *newfildes* are equal, then the action shall ensure that *newfildes* is inherited by the new process with FD_CLOEXEC clear, even if the FD_CLOEXEC flag of *fildes* is set at the time the new process is spawned, and even though *dup2*( ) would not make such a change.

A spawn file actions object is as defined in *posix_spawn_file_actions_addclose*( ).

RETURN VALUE

Upon successful completion, the *posix_spawn_file_actions_adddup2*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_spawn_file_actions_adddup2*( ) function shall fail if:

[EBADF] The value specified by *fildes* or *newfildes* is negative or greater than or equal to {OPEN_MAX}.

[ENOMEM] Insufficient memory exists to add to the spawn file actions object.

The *posix_spawn_file_actions_adddup2*( ) function may fail if:

[EINVAL] The value specified by *file_actions* is invalid.

It shall not be considered an error for the *fildes* argument passed to the *posix_spawn_file_actions_adddup2*( ) function to specify a file descriptor for which the specified operation could not be performed at the time of the call. Any such error shall be detected when the associated file actions object is later used during a *posix_spawn*( ) or *posix_spawnp*( ) operation.

EXAMPLES

None.

APPLICATION USAGE

The *posix_spawn_file_actions_adddup2*( ) function is part of the Spawn option and need not be provided on all implementations.

Implementations may use file descriptors that must be inherited into child processes for the child process to remain conforming, such as for message catalog or tracing purposes. Therefore, an application that calls *posix_spawn_file_actions_adddup2*( ) with an arbitrary integer for *newfildes* risks non-conforming behavior, and this function can only portably be used to overwrite file descriptor values that the application has obtained through explicit actions, or for the three file descriptors corresponding to the standard file streams. In order to avoid a race condition of leaking an unintended file descriptor into a child process or executed program, an application should consider opening all file descriptors with the FD_CLOFORK or FD_CLOEXEC flag, or

both flags, set unless the file descriptor is intended to be inherited by child processes or executed programs, respectively.

RATIONALE

Refer to the RATIONALE section in *posix_spawn_file_actions_addclose( )*.

Although *dup2( )* is required to do nothing when *fildes* and *newfildes* are equal and *fildes* is an open descriptor, the use of *posix_spawn_file_actions_adddup2( )* is required to cause *fildes* to be accessible in the child with FD_CLOEXEC clear. This is because there is no counterpart *posix_spawn_file_actions_fcntl( )* that could be used for clearing the flag as an independent file action. It would also be possible to achieve this effect by using two calls to *posix_spawn_file_actions_adddup2( )* and a temporary *fildes* value known not to conflict with any other file descriptors, coupled with a *posix_spawn_file_actions_close( )* to avoid leaking the temporary, but this approach is complex, and risks [EMFILE] or [ENFILE] failure that can be avoided with the in-place removal of FD_CLOEXEC.

There is no need for *posix_spawn_file_actions_adddup3( )*, since it makes no sense to create a file descriptor with FD_CLOEXEC set before spawning the child process, where that file descriptor would immediately be closed again.

FUTURE DIRECTIONS

None.

SEE ALSO

dup( ), *posix_spawn( )*, *posix_spawn_file_actions_addclose( )*, *posix_spawn_file_actions_destroy( )*

XBD **<spawn.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #104 is applied, noting that the [EBADF] error can apply to the *newfildes* argument in addition to *fildes*.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0440 [149] is applied.

Issue 8

Austin Group Defect 411 is applied, changing requirements relating to the FD_CLOEXEC flag when *fildes* and *newfildes* are equal.

Austin Group Defect 1318 is applied, adding FD_CLOFORK.

NAME

posix_spawn_file_actions_addfchdir — add fchdir action to spawn file actions object (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <spawn.h>

int posix_spawn_file_actions_addfchdir(posix_spawn_file_actions_t
    *file_actions, int fildes);
```

DESCRIPTION

Refer to *posix_spawn_file_actions_addchdir( )*.

NAME

posix_spawn_file_actions_addopen — add open action to spawn file actions object (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <spawn.h>

int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t
    *restrict file_actions, int fildes,
    const char *restrict path, int oflag, mode_t mode);
```

DESCRIPTION

Refer to *posix_spawn_file_actions_addclose( )*.

NAME

posix_spawn_file_actions_destroy, posix_spawn_file_actions_init — destroy and initialize spawn file actions object (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <spawn.h>

int posix_spawn_file_actions_destroy(posix_spawn_file_actions_t
    *file_actions);
int posix_spawn_file_actions_init(posix_spawn_file_actions_t
    *file_actions);
```

DESCRIPTION

The *posix_spawn_file_actions_destroy*( ) function shall destroy the object referenced by *file_actions*; the object becomes, in effect, uninitialized. An implementation may cause *posix_spawn_file_actions_destroy*( ) to set the object referenced by *file_actions* to an invalid value. A destroyed spawn file actions object can be reinitialized using *posix_spawn_file_actions_init*( ); the results of otherwise referencing the object after it has been destroyed are undefined.

The *posix_spawn_file_actions_init*( ) function shall initialize the object referenced by *file_actions* to contain no file actions for *posix_spawn*( ) or *posix_spawnp*( ) to perform.

A spawn file actions object is as defined in *posix_spawn_file_actions_addclose*( ).

The effect of initializing an already initialized spawn file actions object is undefined.

RETURN VALUE

Upon successful completion, these functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_spawn_file_actions_init*( ) function shall fail if:

[ENOMEM] Insufficient memory exists to initialize the spawn file actions object.

The *posix_spawn_file_actions_destroy*( ) function may fail if:

[EINVAL] The value specified by *file_actions* is invalid.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

RATIONALE

Refer to the RATIONALE section in *posix_spawn_file_actions_addclose*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawn_file_actions_addclose*( )

XBD **<spawn.h>**

53771 **CHANGE HISTORY**

53772 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

53773 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

53774 **NAME**

53775 posix_spawnattr_destroy, posix_spawnattr_init — destroy and initialize spawn attributes object
53776 (**ADVANCED REALTIME**)

53777 **SYNOPSIS**

53778 SPN
```
#include <spawn.h>

int posix_spawnattr_destroy(posix_spawnattr_t *attr);
int posix_spawnattr_init(posix_spawnattr_t *attr);
```

53781 **DESCRIPTION**

53782 The *posix_spawnattr_destroy*( ) function shall destroy a spawn attributes object. A destroyed *attr*
53783 attributes object can be reinitialized using *posix_spawnattr_init*( ); the results of otherwise
53784 referencing the object after it has been destroyed are undefined. An implementation may cause
53785 *posix_spawnattr_destroy*( ) to set the object referenced by *attr* to an invalid value.

53786 The *posix_spawnattr_init*( ) function shall initialize a spawn attributes object *attr* with the default
53787 value for all of the individual attributes used by the implementation. Results are undefined if
53788 *posix_spawnattr_init*( ) is called specifying an already initialized *attr* attributes object.

53789 A spawn attributes object is of type **posix_spawnattr_t** (defined in **<spawn.h>**) and is used to
53790 specify the inheritance of process attributes across a spawn operation. POSIX.1-2024 does not
53791 define comparison or assignment operators for the type **posix_spawnattr_t**.

53792 Each implementation shall document the individual attributes it uses and their default values
53793 unless these values are defined by POSIX.1-2024. Attributes not defined by POSIX.1-2024, their
53794 default values, and the names of the associated functions to get and set those attribute values are
53795 implementation-defined.

53796 The resulting spawn attributes object (possibly modified by setting individual attribute values),
53797 is used to modify the behavior of *posix_spawn*( ) or *posix_spawnp*( ). After a spawn attributes
53798 object has been used to spawn a process by a call to a *posix_spawn*( ) or *posix_spawnp*( ), any
53799 function affecting the attributes object (including destruction) shall not affect any process that
53800 has been spawned in this way.

53801 **RETURN VALUE**

53802 Upon successful completion, *posix_spawnattr_destroy*( ) and *posix_spawnattr_init*( ) shall return
53803 zero; otherwise, an error number shall be returned to indicate the error.

53804 **ERRORS**

53805 The *posix_spawnattr_init*( ) function shall fail if:

53806 [ENOMEM] Insufficient memory exists to initialize the spawn attributes object.

53807 The *posix_spawnattr_destroy*( ) function may fail if:

53808 [EINVAL] The value specified by attr is invalid.

53809 **EXAMPLES**

53810 None.

53811 **APPLICATION USAGE**

53812 These functions are part of the Spawn option and need not be provided on all implementations.

53813 **RATIONALE**

53814 The original spawn interface proposed in POSIX.1-2024 defined the attributes that specify the
53815 inheritance of process attributes across a spawn operation as a structure. In order to be able to
53816 separate optional individual attributes under their appropriate options (that is, the *spawn-*
53817 *schedparam* and *spawn-schedpolicy* attributes depending upon the Process Scheduling option), and

also for extensibility and consistency with the newer POSIX interfaces, the attributes interface has been changed to an opaque data type. This interface now consists of the type **posix_spawnattr_t**, representing a spawn attributes object, together with associated functions to initialize or destroy the attributes object, and to set or get each individual attribute. Although the new object-oriented interface is more verbose than the original structure, it is simple to use, more extensible, and easy to implement.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ), *posix_spawnattr_getsigmask*( )

XBD **<spawn.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #106 is applied, noting that the effect of initializing an already initialized spawn attributes option is undefined.

NAME

posix_spawnattr_getflags, posix_spawnattr_setflags — get and set the spawn-flags attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

SPN
```
#include <spawn.h>

int posix_spawnattr_getflags(const posix_spawnattr_t *restrict attr,
    short *restrict flags);
int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
```

DESCRIPTION

The *posix_spawnattr_getflags*( ) function shall obtain the value of the *spawn-flags* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setflags*( ) function shall set the *spawn-flags* attribute in an initialized attributes object referenced by *attr*.

The *spawn-flags* attribute is used to indicate which process attributes are to be changed in the new process image when invoking *posix_spawn*( ) or *posix_spawnp*( ). It is the bitwise-inclusive OR of zero or more of the following flags:

POSIX_SPAWN_RESETIDS
POSIX_SPAWN_SETPGROUP
PS POSIX_SPAWN_SETSCHEDPARAM
POSIX_SPAWN_SETSCHEDULER
POSIX_SPAWN_SETSID
POSIX_SPAWN_SETSIGDEF
POSIX_SPAWN_SETSIGMASK

These flags are defined in **<spawn.h>**. The default value of this attribute shall be as if no flags were set.

RETURN VALUE

Upon successful completion, *posix_spawnattr_getflags*( ) shall return zero and store the value of the *spawn-flags* attribute of *attr* into the object referenced by the *flags* parameter; otherwise, an error number shall be returned to indicate the error.

Upon successful completion, *posix_spawnattr_setflags*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

[EINVAL] The value specified by *attr* is invalid.

The *posix_spawnattr_setflags*( ) function may fail if:

[EINVAL] The value of the attribute being set is not valid.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ), *posix_spawnattr_getsigmask*( )

XBD **<spawn.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 8

Austin Group Defect 1044 is applied, adding POSIX_SPAWN_SETSID.

NAME

posix_spawnattr_getpgroup, posix_spawnattr_setpgroup — get and set the spawn-pgroup attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

SPN
```
#include <spawn.h>

int posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict attr,
    pid_t *restrict pgroup);
int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
```

DESCRIPTION

The *posix_spawnattr_getpgroup*( ) function shall obtain the value of the *spawn-pgroup* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setpgroup*( ) function shall set the *spawn-pgroup* attribute in an initialized attributes object referenced by *attr*.

The *spawn-pgroup* attribute represents the process group to be joined by the new process image in a spawn operation (if POSIX_SPAWN_SETPGROUP is set in the *spawn-flags* attribute). The default value of this attribute shall be zero.

RETURN VALUE

Upon successful completion, *posix_spawnattr_getpgroup*( ) shall return zero and store the value of the *spawn-pgroup* attribute of *attr* into the object referenced by the *pgroup* parameter; otherwise, an error number shall be returned to indicate the error.

Upon successful completion, *posix_spawnattr_setpgroup*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

[EINVAL] The value specified by *attr* is invalid.

The *posix_spawnattr_setpgroup*( ) function may fail if:

[EINVAL] The value of the attribute being set is not valid.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ), *posix_spawnattr_getsigmask*( )

XBD **<spawn.h>**

53928 **CHANGE HISTORY**

53929 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

NAME

posix_spawnattr_getschedparam, posix_spawnattr_setschedparam — get and set the spawn-schedparam attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

SPN PS

```
#include <spawn.h>
#include <sched.h>

int posix_spawnattr_getschedparam(const posix_spawnattr_t
    *restrict attr, struct sched_param *restrict schedparam);
int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
    const struct sched_param *restrict schedparam);
```

DESCRIPTION

The *posix_spawnattr_getschedparam*( ) function shall obtain the value of the *spawn-schedparam* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setschedparam*( ) function shall set the *spawn-schedparam* attribute in an initialized attributes object referenced by *attr*.

The *spawn-schedparam* attribute represents the scheduling parameters to be assigned to the new process image in a spawn operation (if POSIX_SPAWN_SETSCHEDULER or POSIX_SPAWN_SETSCHEDPARAM is set in the *spawn-flags* attribute). The default value of this attribute is unspecified.

RETURN VALUE

Upon successful completion, *posix_spawnattr_getschedparam*( ) shall return zero and store the value of the *spawn-schedparam* attribute of *attr* into the object referenced by the *schedparam* parameter; otherwise, an error number shall be returned to indicate the error.

Upon successful completion, *posix_spawnattr_setschedparam*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

[EINVAL] The value specified by *attr* is invalid.

The *posix_spawnattr_setschedparam*( ) function may fail if:

[EINVAL] The value of the attribute being set is not valid.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn and Process Scheduling options and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedpolicy*( ), *posix_spawnattr_getsigmask*( )

53973 XBD **<sched.h>**, **<spawn.h>**

53974 **CHANGE HISTORY**

53975 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

NAME

posix_spawnattr_getschedpolicy, posix_spawnattr_setschedpolicy — get and set the spawn-schedpolicy attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

SPN PS

```
#include <spawn.h>
#include <sched.h>

int posix_spawnattr_getschedpolicy(const posix_spawnattr_t
    *restrict attr, int *restrict schedpolicy);
int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
    int schedpolicy);
```

DESCRIPTION

The *posix_spawnattr_getschedpolicy*( ) function shall obtain the value of the *spawn-schedpolicy* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setschedpolicy*( ) function shall set the *spawn-schedpolicy* attribute in an initialized attributes object referenced by *attr*.

The *spawn-schedpolicy* attribute represents the scheduling policy to be assigned to the new process image in a spawn operation (if POSIX_SPAWN_SETSCHEDULER is set in the *spawn-flags* attribute). The default value of this attribute is unspecified.

RETURN VALUE

Upon successful completion, *posix_spawnattr_getschedpolicy*( ) shall return zero and store the value of the *spawn-schedpolicy* attribute of *attr* into the object referenced by the *schedpolicy* parameter; otherwise, an error number shall be returned to indicate the error.

Upon successful completion, *posix_spawnattr_setschedpolicy*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

[EINVAL] The value specified by *attr* is invalid.

The *posix_spawnattr_setschedpolicy*( ) function may fail if:

[EINVAL] The value of the attribute being set is not valid.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn and Process Scheduling options and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getsigmask*( )

XBD **<sched.h>**, **<spawn.h>**

54019 **CHANGE HISTORY**

54020 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

NAME

posix_spawnattr_getsigdefault, posix_spawnattr_setsigdefault — get and set the spawn-sigdefault attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <signal.h>
#include <spawn.h>

int posix_spawnattr_getsigdefault(const posix_spawnattr_t
    *restrict attr, sigset_t *restrict sigdefault);
int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
    const sigset_t *restrict sigdefault);
```

DESCRIPTION

The *posix_spawnattr_getsigdefault*( ) function shall obtain the value of the *spawn-sigdefault* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setsigdefault*( ) function shall set the *spawn-sigdefault* attribute in an initialized attributes object referenced by *attr*.

The *spawn-sigdefault* attribute represents the set of signals to be forced to default signal handling in the new process image (if POSIX_SPAWN_SETSIGDEF is set in the *spawn-flags* attribute) by a spawn operation. The default value of this attribute shall be an empty signal set.

RETURN VALUE

Upon successful completion, *posix_spawnattr_getsigdefault*( ) shall return zero and store the value of the *spawn-sigdefault* attribute of *attr* into the object referenced by the *sigdefault* parameter; otherwise, an error number shall be returned to indicate the error.

Upon successful completion, *posix_spawnattr_setsigdefault*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

[EINVAL] The value specified by *attr* is invalid.

The *posix_spawnattr_setsigdefault*( ) function may fail if:

[EINVAL] The value of the attribute being set is not valid.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ), *posix_spawnattr_getsigmask*( )

XBD **<signal.h>**, **<spawn.h>**

54062 **CHANGE HISTORY**

54063 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

NAME

posix_spawnattr_getsigmask, posix_spawnattr_setsigmask — get and set the spawn-sigmask attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <signal.h>
#include <spawn.h>

int posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict attr,
    sigset_t *restrict sigmask);
int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,
    const sigset_t *restrict sigmask);
```

DESCRIPTION

The *posix_spawnattr_getsigmask*( ) function shall obtain the value of the *spawn-sigmask* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setsigmask*( ) function shall set the *spawn-sigmask* attribute in an initialized attributes object referenced by *attr*.

The *spawn-sigmask* attribute represents the signal mask in effect in the new process image of a spawn operation (if POSIX_SPAWN_SETSIGMASK is set in the *spawn-flags* attribute). The default value of this attribute is unspecified.

RETURN VALUE

Upon successful completion, *posix_spawnattr_getsigmask*( ) shall return zero and store the value of the *spawn-sigmask* attribute of *attr* into the object referenced by the *sigmask* parameter; otherwise, an error number shall be returned to indicate the error.

Upon successful completion, *posix_spawnattr_setsigmask*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

[EINVAL] The value specified by *attr* is invalid.

The *posix_spawnattr_setsigmask*( ) function may fail if:

[EINVAL] The value of the attribute being set is not valid.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( )

XBD **<signal.h>**, **<spawn.h>**

54106 **CHANGE HISTORY**

54107 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

NAME

posix_spawnattr_init — initialize the spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <spawn.h>

int posix_spawnattr_init(posix_spawnattr_t *attr);
```

DESCRIPTION

Refer to *posix_spawnattr_destroy( )*.

NAME

posix_spawnattr_setflags — set the spawn-flags attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <spawn.h>

int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
```

DESCRIPTION

Refer to *posix_spawnattr_getflags( )*.

NAME

posix_spawnattr_setpgroup — set the spawn-pgroup attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <spawn.h>

int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
```

DESCRIPTION

Refer to *posix_spawnattr_getpgroup( )*.

NAME

posix_spawnattr_setschedparam — set the spawn-schedparam attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

SPN PS

```
#include <sched.h>
#include <spawn.h>

int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
    const struct sched_param *restrict schedparam);
```

DESCRIPTION

Refer to *posix_spawnattr_getschedparam( )*.

NAME

posix_spawnattr_setschedpolicy — set the spawn-schedpolicy attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

SPN PS

```
#include <sched.h>
#include <spawn.h>

int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
    int schedpolicy);
```

DESCRIPTION

Refer to *posix_spawnattr_getschedpolicy*( ).

NAME

posix_spawnattr_setsigdefault — set the spawn-sigdefault attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <signal.h>
#include <spawn.h>

int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
    const sigset_t *restrict sigdefault);
```

DESCRIPTION

Refer to *posix_spawnattr_getsigdefault( )*.

NAME

posix_spawnattr_setsigmask — set the spawn-sigmask attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <signal.h>
#include <spawn.h>

int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,
    const sigset_t *restrict sigmask);
```

DESCRIPTION

Refer to *posix_spawnattr_getsigmask( )*.

NAME

posix_spawnp — spawn a process (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <spawn.h>

int posix_spawnp(pid_t *restrict pid, const char *restrict file,
    const posix_spawn_file_actions_t *file_actions,
    const posix_spawnattr_t *restrict attrp,
    char *const argv[restrict], char *const envp[restrict]);
```

DESCRIPTION

Refer to *posix_spawn( )*.

NAME

posix_typed_mem_get_info — query typed memory information (**ADVANCED REALTIME**)

SYNOPSIS

TYM
```
#include <sys/mman.h>

int posix_typed_mem_get_info(int fildes,
    struct posix_typed_mem_info *info);
```

DESCRIPTION

The *posix_typed_mem_get_info*( ) function shall return, in the *posix_tmi_length* field of the **posix_typed_mem_info** structure pointed to by *info*, the maximum length which may be successfully allocated by the typed memory object designated by *fildes*. This maximum length shall take into account the flag POSIX_TYPED_MEM_ALLOCATE or POSIX_TYPED_MEM_ALLOCATE_CONTIG specified when the typed memory object represented by *fildes* was opened. The maximum length is dynamic; therefore, the value returned is valid only while the current mapping of the corresponding typed memory pool remains unchanged.

If *fildes* represents a typed memory object opened with neither the POSIX_TYPED_MEM_ALLOCATE flag nor the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag specified, the returned value of *info*->*posix_tmi_length* is unspecified.

The *posix_typed_mem_get_info*( ) function may return additional implementation-defined information in other fields of the **posix_typed_mem_info** structure pointed to by *info*.

If the memory object specified by *fildes* is not a typed memory object, then the behavior of this function is undefined.

RETURN VALUE

Upon successful completion, the *posix_typed_mem_get_info*( ) function shall return zero; otherwise, the corresponding error status value shall be returned.

ERRORS

The *posix_typed_mem_get_info*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid open file descriptor.

[ENODEV] The *fildes* argument is not connected to a memory object supported by this function.

This function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

An application that needs to allocate a block of typed memory with length dependent upon the amount of memory currently available must either query the typed memory object to obtain the amount available, or repeatedly invoke *mmap*( ) attempting to guess an appropriate length. While the latter method is existing practice with *malloc*( ), it is awkward and imprecise. The *posix_typed_mem_get_info*( ) function allows an application to immediately determine available memory. This is particularly important for typed memory objects that may in some cases be scarce resources. Note that when a typed memory pool is a shared resource, some form of mutual-exclusion or synchronization may be required while typed memory is being queried and

allocated to prevent race conditions.

The existing *fstat*( ) function is not suitable for this purpose. We realize that implementations may wish to provide other attributes of typed memory objects (for example, alignment requirements, page size, and so on). The *fstat*( ) function returns a structure which is not extensible and, furthermore, contains substantial information that is inappropriate for typed memory objects.

FUTURE DIRECTIONS

None.

SEE ALSO

fstat( ), *mmap*( ), *posix_typed_mem_open*( )

XBD **<sys/mman.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

NAME

posix_typed_mem_open — open a typed memory object (**ADVANCED REALTIME**)

SYNOPSIS

TYM
```
#include <sys/mman.h>

int posix_typed_mem_open(const char *name, int oflag, int tflag);
```

DESCRIPTION

The *posix_typed_mem_open*( ) function shall establish a connection between the typed memory object specified by the string pointed to by *name* and a file descriptor. It shall create an open file description that refers to the typed memory object and a file descriptor that refers to that open file description. The file descriptor shall be allocated as described in Section 2.6 (on page 525) and can be used by other functions to refer to that typed memory object. It is unspecified whether the name appears in the file system and is visible to other functions that take pathnames as arguments. The *name* argument conforms to the construction rules for a pathname, except that the interpretation of <slash> characters other than the leading <slash> character in *name* is implementation-defined, and that the length limits for the *name* argument are implementation-defined and need not be the same as the pathname limits {PATH_MAX} and {NAME_MAX}. If *name* begins with the <slash> character, then processes calling *posix_typed_mem_open*( ) with the same value of *name* shall refer to the same typed memory object. If *name* does not begin with the <slash> character, the effect is implementation-defined.

Each typed memory object supported in a system shall be identified by a name which specifies not only its associated typed memory pool, but also the path or port by which it is accessed. That is, the same typed memory pool accessed via several different ports shall have several different corresponding names. The binding between names and typed memory objects is established in an implementation-defined manner. Unlike shared memory objects, there is no way within POSIX.1-2024 for a program to create a typed memory object.

The value of *tflag* shall determine how the typed memory object behaves when subsequently mapped by calls to *mmap*( ). At most, one of the following flags defined in **<sys/mman.h>** may be specified:

POSIX_TYPED_MEM_ALLOCATE
: Allocate on *mmap*( ).

POSIX_TYPED_MEM_ALLOCATE_CONTIG
: Allocate contiguously on *mmap*( ).

POSIX_TYPED_MEM_MAP_ALLOCATABLE
: Map on *mmap*( ), without affecting allocatability.

If *tflag* has the flag POSIX_TYPED_MEM_ALLOCATE specified, any subsequent call to *mmap*( ) using the returned file descriptor shall result in allocation and mapping of typed memory from the specified typed memory pool. The allocated memory may be a contiguous previously unallocated area of the typed memory pool or several non-contiguous previously unallocated areas (mapped to a contiguous portion of the process address space). If *tflag* has the flag POSIX_TYPED_MEM_ALLOCATE_CONTIG specified, any subsequent call to *mmap*( ) using the returned file descriptor shall result in allocation and mapping of a single contiguous previously unallocated area of the typed memory pool (also mapped to a contiguous portion of the process address space). If *tflag* has none of the flags POSIX_TYPED_MEM_ALLOCATE or POSIX_TYPED_MEM_ALLOCATE_CONTIG specified, any subsequent call to *mmap*( ) using the returned file descriptor shall map an application-chosen area from the specified typed memory pool such that this mapped area becomes unavailable for allocation until unmapped by all processes. If *tflag* has the flag POSIX_TYPED_MEM_MAP_ALLOCATABLE specified, any

subsequent call to *mmap*( ) using the returned file descriptor shall map an application-chosen area from the specified typed memory pool without an effect on the availability of that area for allocation; that is, mapping such an object leaves each byte of the mapped area unallocated if it was unallocated prior to the mapping or allocated if it was allocated prior to the mapping. Appropriate privileges to specify the POSIX_TYPED_MEM_MAP_ALLOCATABLE flag are implementation-defined.

If successful, *posix_typed_mem_open*( ) shall return a file descriptor for the typed memory object. The open file description is new, and therefore the file descriptor shall not share it with any other processes. It is unspecified whether the file offset is set. The FD_CLOEXEC file descriptor flag associated with the new file descriptor shall be cleared unless *oflag* includes O_CLOEXEC. The FD_CLOFORK file descriptor flag associated with the new file descriptor shall be cleared unless *oflag* includes O_CLOFORK.

The behavior of *msync*( ), *ftruncate*( ), and all file operations other than *mmap*( ), *posix_mem_offset*( ), *posix_typed_mem_get_info*( ), *fstat*( ), *dup*( ), *dup2*( ), *dup3*( ), and *close*( ), is unspecified when passed a file descriptor connected to a typed memory object by this function.

The file status flags of the open file description shall be set according to the value of *oflag*. Applications shall specify exactly one of the three access mode values described below as the value of *oflag*.

O_RDONLY Open for read access only.

O_WRONLY Open for write access only.

O_RDWR Open for read or write access.

Additionally, the value of *oflag* can include the following flags:

O_CLOEXEC Set the FD_CLOEXEC file descriptor flag.

O_CLOFORK Set the FD_CLOFORK file descriptor flag.

RETURN VALUE

Upon successful completion, the *posix_typed_mem_open*( ) function shall return a non-negative integer representing the file descriptor. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *posix_typed_mem_open*( ) function shall fail if:

[EACCES] The typed memory object exists and the permissions specified by *oflag* are denied.

[EINTR] The *posix_typed_mem_open*( ) operation was interrupted by a signal.

[EINVAL] The flags specified in *tflag* are invalid (more than one of POSIX_TYPED_MEM_ALLOCATE, POSIX_TYPED_MEM_ALLOCATE_CONTIG, or POSIX_TYPED_MEM_MAP_ALLOCATABLE is specified).

[EMFILE] All file descriptors available to the process are currently open.

[ENFILE] Too many file descriptors are currently open in the system.

[ENOENT] The named typed memory object does not exist.

[EPERM] The caller lacks appropriate privileges to specify the POSIX_TYPED_MEM_MAP_ALLOCATABLE flag in the *tflag* argument.

The *posix_typed_mem_open*( ) function may fail if:

[ENAMETOOLONG]

The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI systems, or has a pathname component that is longer than {_POSIX_NAME_MAX} on systems that do not support the XSI option or longer than {_XOPEN_NAME_MAX} on XSI systems.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The use of the O_CLOEXEC and O_CLOFORK flags to *posix_typed_mem_open*( ) is necessary to avoid leaking typed memory file descriptors to child processes, since *fcntl*( ) has unspecified results on typed memory objects and therefore cannot be used to set FD_CLOEXEC or FD_CLOFORK after the file descriptor has been opened. The *exec* family of functions already unmaps all memory associated with a typed memory object, but does not close the file descriptor unless FD_CLOEXEC is also set.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.6 (on page 525), *close*( ), *dup*( ), *exec*, *fcntl*( ), *fstat*( ), *ftruncate*( ), *mmap*( ), *msync*( ), *posix_mem_offset*( ), *posix_typed_mem_get_info*( ), *umask*( )

XBD **<fcntl.h>**, **<sys/mman.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0442 [119,428] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0256 [835], XSH/TC2-2008/0257 [835], and XSH/TC2-2008/0258 [835] are applied.

Issue 8

Austin Group Defects 411 and 1318 are applied, adding O_CLOEXEC, O_CLOFORK, and *dup3*( ).

Austin Group Defect 593 is applied, removing a reference to **<fcntl.h>** from the DESCRIPTION section.

NAME

pow, powf, powl — power function

SYNOPSIS

```
#include <math.h>

double pow(double x, double y);
float powf(float x, float y);
long double powl(long double x, long double y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the value of x raised to the power y, x^y. If x is negative, the application shall ensure that y is an integer value.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the value of x raised to the power y.

MX For finite values of $x < 0$, and finite non-integer values of y, a domain error shall occur and either a NaN (if representable), or an implementation-defined value shall be returned.

If the correct value would cause overflow, a range error shall occur and *pow*( ), *powf*( ), and *powl*( ) shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL, respectively, with the same sign as the correct value of the function.

MXX If the correct value would cause underflow, and is not representable, a range error may occur, and *pow*( ), *powf*( ), and *powl*( ) shall return 0.0, or (if IEC 60559 Floating-Point is not supported) an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

CX For $y < 0$, if x is zero, a pole error may occur and *pow*( ), *powf*( ), and *powl*( ) shall return
MX ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL, respectively. On systems that support the IEC 60559 Floating-Point option, if x is ±0:

- If y is an odd integer, a pole error shall occur and *pow*( ), *powf*( ), and *powl*( ) shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL, respectively.
- If y is finite and is not an odd integer, a pole error shall occur and *pow*( ), *powf*( ), and *powl*( ) shall return HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.
- If y is –Inf, a pole error may occur and *pow*( ), *powf*( ), and *powl*( ) shall return HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

MX If x or y is a NaN, a NaN shall be returned (unless specified elsewhere in this description).

For any value of y (including NaN), if x is +1, 1.0 shall be returned.

For any value of x (including NaN), if y is ±0, 1.0 shall be returned.

For any odd integer value of $y > 0$, if x is ±0, ±0 shall be returned.

For $y > 0$ and not an odd integer, if x is ±0, +0 shall be returned.

If x is –1, and y is ±Inf, 1.0 shall be returned.

For $|x| < 1$, if y is –Inf, +Inf shall be returned.

For $|x| > 1$, if y is –Inf, +0 shall be returned.

For $|x| < 1$, if y is +Inf, +0 shall be returned.

For $|x| > 1$, if y is +Inf, +Inf shall be returned.

For y an odd integer < 0, if x is –Inf, –0 shall be returned.

For $y < 0$ and not an odd integer, if x is –Inf, +0 shall be returned.

For y an odd integer > 0, if x is –Inf, –Inf shall be returned.

For $y > 0$ and not an odd integer, if x is –Inf, +Inf shall be returned.

For $y < 0$, if x is +Inf, +0 shall be returned.

For $y > 0$, if x is +Inf, +Inf shall be returned.

MXX If the correct value would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

ERRORS

These functions shall fail if:

Domain Error — The value of x is negative and y is a finite non-integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

MX Pole Error — The value of x is zero and y is negative.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

Range Error — The result overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

Pole Error — The value of x is zero and y is negative.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

Range Error — The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow

floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exp( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

The *powf*( ) and *powl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/42 is applied, correcting the third paragraph in the RETURN VALUE section.

Issue 7

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #51 (SD5-XSH-ERN-81) is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0443 [68], XSH/TC1-2008/0444 [148], and XSH/TC1-2008/0445 [68] are applied.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018 standard.

54479 **NAME**

54480 ppoll — input/output multiplexing

54481 **SYNOPSIS**

54482 ```
#include <poll.h>
```

54483 ```
int ppoll(struct pollfd fds[], nfds_t nfds,
54484         const struct timespec *restrict timeout,
54485         const sigset_t *restrict sigmask);
```

54486 **DESCRIPTION**

54487 Refer to *poll( )*.

54488 **NAME**

54489 pread — read from a file

54490 **SYNOPSIS**

54491 `#include <unistd.h>`

54492 `ssize_t pread(int` *`fildes`*`, void *`*`buf`*`, size_t` *`nbyte`*`, off_t` *`offset`*`);`

54493 **DESCRIPTION**

54494 Refer to *read( )*.

54495 **NAME**

54496 printf — print formatted output

54497 **SYNOPSIS**

54498 `#include <stdio.h>`

54499 `int printf(const char *restrict format, ...);`

54500 **DESCRIPTION**

54501 Refer to *fprintf( )*.

NAME

pselect, select — synchronous I/O multiplexing

SYNOPSIS

```
#include <sys/select.h>

int pselect(int nfds, fd_set *restrict readfds,
    fd_set *restrict writefds, fd_set *restrict errorfds,
    const struct timespec *restrict timeout,
    const sigset_t *restrict sigmask);
int select(int nfds, fd_set *restrict readfds,
    fd_set *restrict writefds, fd_set *restrict errorfds,
    struct timeval *restrict timeout);
void FD_CLR(int fd, fd_set *fdset);
int FD_ISSET(int fd, const fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_ZERO(fd_set *fdset);
```

DESCRIPTION

The *pselect*( ) function shall examine the file descriptor sets whose addresses are passed in the *readfds*, *writefds*, and *errorfds* parameters to see whether some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively.

The *select*( ) function shall be equivalent to the *pselect*( ) function, except as follows:

- For the *select*( ) function, the timeout period is given in seconds and microseconds in an argument of type **struct timeval**, whereas for the *pselect*( ) function the timeout period is given in seconds and nanoseconds in an argument of type **struct timespec**.
- The *select*( ) function has no *sigmask* argument; it shall behave as *pselect*( ) does when *sigmask* is a null pointer.
- Upon successful completion, the *select*( ) function may modify the object pointed to by the *timeout* argument.

The *pselect*( ) and *select*( ) functions shall support regular files, terminal and pseudo-terminal devices, FIFOs, pipes, and sockets. The behavior of *pselect*( ) and *select*( ) on file descriptors that refer to other types of file is unspecified.

The *nfds* argument specifies the range of descriptors to be tested. The first *nfds* descriptors shall be checked in each set; that is, the descriptors from zero through *nfds*–1 in the descriptor sets shall be examined.

If the *readfds* argument is not a null pointer, it points to an object of type **fd_set** that on input specifies the file descriptors to be checked for being ready to read, and on output indicates which file descriptors are ready to read.

If the *writefds* argument is not a null pointer, it points to an object of type **fd_set** that on input specifies the file descriptors to be checked for being ready to write, and on output indicates which file descriptors are ready to write.

If the *errorfds* argument is not a null pointer, it points to an object of type **fd_set** that on input specifies the file descriptors to be checked for error conditions pending, and on output indicates which file descriptors have error conditions pending.

Upon successful completion, the *pselect*( ) or *select*( ) function shall modify the objects pointed to by the *readfds*, *writefds*, and *errorfds* arguments to indicate which file descriptors are ready for reading, ready for writing, or have an error condition pending, respectively, and shall return the total number of ready descriptors in all the output sets. For each file descriptor less than *nfds*, the

corresponding bit shall be set upon successful completion if it was set on input and the associated condition is true for that file descriptor.

If none of the selected descriptors are ready for the requested operation, the *pselect*( ) or *select*( ) function shall block until at least one of the requested operations becomes ready, until the *timeout* occurs, or until interrupted by a signal. The *timeout* parameter controls how long the *pselect*( ) or *select*( ) function shall take before timing out. If the *timeout* parameter is not a null pointer, it specifies a maximum interval to wait for the selection to complete. If the specified time interval expires without any requested operation becoming ready, the function shall return. If the *timeout* parameter is a null pointer, then the call to *pselect*( ) or *select*( ) shall block indefinitely until at least one descriptor meets the specified criteria. To effect a poll, the *timeout* parameter should not be a null pointer, and should point to a zero-valued **timespec** structure.

The use of a timeout does not affect any pending timers set up by *alarm*( ).

Implementations may place limitations on the maximum timeout interval supported. All implementations shall support a maximum timeout interval of at least 31 days. If the *timeout* argument specifies a timeout interval greater than the implementation-defined maximum value, the maximum value shall be used as the actual timeout value. Implementations may also place limitations on the granularity of timeout intervals. If the requested timeout interval requires a finer granularity than the implementation supports, the actual timeout interval shall be rounded up to the next supported value.

If *sigmask* is not a null pointer, then the *pselect*( ) function shall replace the signal mask of the caller by the set of signals pointed to by *sigmask* before examining the descriptors, and shall restore the signal mask of the calling thread before returning. If a signal is unmasked as a result of the signal mask being altered by *pselect*( ), and a signal-catching function is called for that signal during the execution of the *pselect*( ) function, and SA_RESTART is clear for the interrupting signal, then

- If none of the selected file descriptors are ready, *pselect*( ) shall immediately fail with the [EINTR] error after the signal-catching function returns.
- If one or more of the selected file descriptors are ready, it is unspecified whether *pselect*( ) behaves the same as if none of the descriptors were ready (failing with [EINTR] as above) or behaves the same as if it was not interrupted (returning the total number of ready descriptors).

A descriptor shall be considered ready for reading when a call to an input function with O_NONBLOCK clear would not block, whether or not the function would transfer data successfully. (The function might return data, an end-of-file indication, or an error other than one indicating that it is blocked, and in each of these cases the descriptor shall be considered ready for reading.)

A descriptor shall be considered ready for writing when a call to an output function with O_NONBLOCK clear would not block, whether or not the function would transfer data successfully.

If a socket has a pending error, it shall be considered to have an exceptional condition pending. Otherwise, what constitutes an exceptional condition is file type-specific. For a file descriptor for use with a socket, it is protocol-specific except as noted below. For other file types it is implementation-defined. If the operation is meaningless for a particular file type, *pselect*( ) or *select*( ) shall indicate that the descriptor is ready for read or write operations, and shall indicate that the descriptor has no exceptional condition pending.

If a descriptor refers to a socket, the implied input function is the *recvmsg*( ) function with parameters requesting normal and ancillary data, such that the presence of either type shall

cause the socket to be marked as readable. The presence of out-of-band data shall be checked if the socket option SO_OOBINLINE has been enabled, as out-of-band data is enqueued with normal data. If the socket is currently listening, then it shall be marked as readable if an incoming connection request has been received, and a call to the *accept*( ) or *accept4*( ) function shall complete without blocking.

If a descriptor refers to a socket, the implied output function is the *sendmsg*( ) function supplying an amount of normal data equal to the current value of the SO_SNDLOWAT option for the socket. If a non-blocking call to the *connect*( ) function has been made for a socket, and the connection attempt has either succeeded or failed leaving a pending error, the socket shall be marked as writable.

A socket shall be considered to have an exceptional condition pending if a receive operation with O_NONBLOCK clear for the open file description and with the MSG_OOB flag set would return out-of-band data without blocking. (It is protocol-specific whether the MSG_OOB flag would be used to read out-of-band data.) A socket shall also be considered to have an exceptional condition pending if an out-of-band data mark is present in the receive queue. Other circumstances under which a socket may be considered to have an exceptional condition pending are protocol-specific and implementation-defined.

If the *readfds*, *writefds*, and *errorfds* arguments are all null pointers and the *timeout* argument is not a null pointer, the *pselect*( ) or *select*( ) function shall block for the time specified, or until interrupted by a signal. If the *readfds*, *writefds*, and *errorfds* arguments are all null pointers and the *timeout* argument is a null pointer, the *pselect*( ) or *select*( ) function shall block until interrupted by a signal.

File descriptors associated with regular files shall always select true for ready to read, ready to write, and error conditions.

On failure, the objects pointed to by the *readfds*, *writefds*, and *errorfds* arguments shall not be modified. If the timeout interval expires without the specified condition being true for any of the specified file descriptors, the objects pointed to by the *readfds*, *writefds*, and *errorfds* arguments shall have all bits set to 0.

File descriptor masks of type **fd_set** can be initialized and tested with *FD_CLR*( ), *FD_ISSET*( ), *FD_SET*( ), and *FD_ZERO*( ). It is unspecified whether each of these is a macro or a function. If a macro definition is suppressed in order to access an actual function, or a program defines an external identifier with any of these names, the behavior is undefined.

FD_CLR(*fd*, *fdsetp*) shall remove the file descriptor *fd* from the set pointed to by *fdsetp*. If *fd* is not a member of this set, there shall be no effect on the set, and this shall not be treated as an error.

FD_ISSET(*fd*, *fdsetp*) shall evaluate to non-zero if the file descriptor *fd* is a member of the set pointed to by *fdsetp*, and shall evaluate to zero otherwise.

FD_SET(*fd*, *fdsetp*) shall add the file descriptor *fd* to the set pointed to by *fdsetp*. If the file descriptor *fd* is already in this set, there shall be no effect on the set, and this shall not be treated as an error.

FD_ZERO(*fdsetp*) shall initialize the descriptor set pointed to by *fdsetp* to the null set. No error is returned if the set is not empty at the time *FD_ZERO*( ) is invoked.

The behavior of these macros is undefined if the *fd* argument is less than 0 or greater than or equal to FD_SETSIZE, or if *fd* is not a valid file descriptor, or if any of the arguments are expressions with side-effects.

If a thread gets canceled during a *pselect*( ) call, the signal mask in effect when executing the registered cleanup functions is either the original signal mask or the signal mask installed as part

of the *pselect*( ) call.

RETURN VALUE

Upon successful completion, the *pselect*( ) and *select*( ) functions shall return the total number of bits set in the bit masks. Otherwise, –1 shall be returned, and *errno* shall be set to indicate the error.

FD_CLR( ), *FD_SET*( ), and *FD_ZERO*( ) do not return a value. *FD_ISSET*( ) shall return a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set pointed to by *fdset*, and 0 otherwise.

ERRORS

Under the following conditions, *pselect*( ) and *select*( ) shall fail and set *errno* to:

[EBADF] One or more of the file descriptor sets specified a file descriptor that is not a valid open file descriptor.

[EINTR] The function was interrupted by a signal.

If SA_RESTART has been set for the interrupting signal, it is implementation-defined whether the function restarts or returns with [EINTR].

[EINVAL] An invalid timeout interval was specified.

[EINVAL] The *nfds* argument is less than 0 or greater than FD_SETSIZE.

EXAMPLES

None.

APPLICATION USAGE

The use of *select*( ) and *pselect*( ) requires that the application construct the set of file descriptors to work on each time through a polling loop, and is inherently limited from operating on file descriptors larger than FD_SETSIZE. Also, the amount of work to perform scales as *nfds* increases, even if the number of file descriptors selected within the larger set remains the same. Thus, applications may wish to consider using *poll*( ) and *ppoll*( ) instead, for better scaling.

When a *pselect*( ) or *select*( ) call indicates a file descriptor is ready for reading, this means that if an attempt to read data had been made at the time that the status of the file descriptor was checked, it would have returned at least one byte of data, an end-of-file indication, or an error, without blocking (even if O_NONBLOCK is clear). When a *pselect*( ) or *select*( ) call indicates that a file descriptor is ready for writing, this means that if an attempt to write one byte of data had been made at the time that the status of the file descriptor was checked, it would have written that byte or returned an error, without blocking. However, if an attempt to write more than one byte had been made, it might have blocked (if O_NONBLOCK is clear). In both cases, by the time the call returns and a subsequent I/O operation is attempted, the state of the file descriptor might have changed (for example, because another thread read or wrote some data) and, if O_NONBLOCK is clear, there is no guarantee that the operation will not block (unless it would not block for some other reason, such as setting MIN=0 and TIME=0 for a terminal in non-canonical mode). Therefore it is recommended that applications always set O_NONBLOCK on file descriptors whose readiness for I/O they query with *pselect*( ) or *select*( ).

RATIONALE

In earlier versions of the Single UNIX Specification, the *select*( ) function was defined in the **<sys/time.h>** header. This is now changed to **<sys/select.h>**. The rationale for this change was as follows: the introduction of the *pselect*( ) function included the **<sys/select.h>** header and the **<sys/select.h>** header defines all the related definitions for the *pselect*( ) and *select*( ) functions. Backwards-compatibility to existing XSI implementations is handled by allowing **<sys/time.h>** to include **<sys/select.h>**.

Code which wants to avoid the ambiguity of the signal mask for thread cancellation handlers can install an additional cancellation handler which resets the signal mask to the expected value.

```
void cleanup(void *arg)
{
    sigset_t *ss = (sigset_t *) arg;
    pthread_sigmask(SIG_SETMASK, ss, NULL);
}

int call_pselect(int nfds, fd_set *readfds, fd_set *writefds,
    fd_set errorfds, const struct timespec *timeout,
    const sigset_t *sigmask)
{
    sigset_t oldmask;
    int result;
    pthread_sigmask(SIG_SETMASK, NULL, &oldmask);
    pthread_cleanup_push(cleanup, &oldmask);
    result = pselect(nfds, readfds, writefds, errorfds, timeout, sigmask);
    pthread_cleanup_pop(0);
    return result;
}
```

FUTURE DIRECTIONS

None.

SEE ALSO

accept( ), *alarm*( ), *connect*( ), *fcntl*( ), *poll*( ), *read*( ), *recvmsg*( ), *sendmsg*( ), *write*( )

XBD **<sys/select.h>**, **<sys/time.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

In the ERRORS section, the text has been changed to indicate that [EINVAL] is returned when *nfds* is less than 0 or greater than FD_SETSIZE. It previously stated less than 0, or greater than or equal to FD_SETSIZE.

Text about *timeout* is moved from the APPLICATION USAGE section to the DESCRIPTION.

Issue 6

The Open Group Corrigendum U026/6 is applied, changing the occurrences of *readfs* and *writefs* in the *select*( ) DESCRIPTION to be *readfds* and *writefds*.

Text referring to sockets is added to the DESCRIPTION.

The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are marked as part of the XSI STREAMS Option Group.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- These functions are now mandatory.

The *pselect*( ) function is added for alignment with IEEE Std 1003.1g-2000 and additional detail related to sockets semantics is added to the DESCRIPTION.

The *select*( ) function now requires inclusion of **<sys/select.h>**.

The **restrict** keyword is added to the *select*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/70 is applied, updating the DESCRIPTION to reference the signal mask in terms of the calling thread rather than the process.

Issue 7

SD5-XSH-ERN-122 is applied, adding text to the DESCRIPTION for when a thread is canceled during a call to *pselect*( ), and adding example code to the RATIONALE.

Functionality relating to the XSI STREAMS option is marked obsolescent.

Functionality relating to the Threads option is moved to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0446 [372] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0259 [680] is applied.

Issue 8

Austin Group Defect 220 is applied, adding `const` to the second parameter of *FD_ISSET*( ).

Austin Group Defect 411 is applied, adding *accept4*( ).

Austin Group Defect 1186 is applied, clarifying the behavior when the *pselect*( ) function is interrupted by a signal.

Austin Group Defect 1263 is applied, changing the APPLICATION USAGE section.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1448 is applied, changing the APPLICATION USAGE section.

NAME

psiginfo, psignal — write signal information to standard error

SYNOPSIS

CX
```
#include <signal.h>

void psiginfo(const siginfo_t *pinfo, const char *message);
void psignal(int signum, const char *message);
```

DESCRIPTION

The *psiginfo*( ) and *psignal*( ) functions shall write a language-dependent message associated with a signal number to the standard error stream as follows:

- First, if *message* is not a null pointer and is not the empty string, the string pointed to by the *message* argument shall be written, followed by a <colon> and a <space>.
- Then the signal description string associated with *signum* or with the signal indicated by *pinfo* shall be written, followed by a <newline>.

For *psiginfo*( ), the application shall ensure that the argument *pinfo* references a valid **siginfo_t** structure. For *psignal*( ), if *signum* is not a valid signal number, the behavior is implementation-defined.

The *psiginfo*( ) and *psignal*( ) functions shall not change the orientation of the standard error stream.

The *psiginfo*( ) and *psignal*( ) functions shall mark for update the last data modification and last file status change timestamps of the file associated with the standard error stream at some time between their successful completion and *exit*( ), *abort*( ), or the completion of *fflush*( ) or *fclose*( ) on *stderr*.

The *psiginfo*( ) and *psignal*( ) functions shall not change the setting of *errno* if successful.

On error, the *psiginfo*( ) and *psignal*( ) functions shall set the error indicator for the stream to which *stderr* points, and shall set *errno* to indicate the error.

Since no value is returned, an application wishing to check for error situations should set *errno* to 0, then call *psiginfo*( ) or *psignal*( ), then check *errno*.

RETURN VALUE

These functions shall not return a value.

ERRORS

Refer to *fputc*( ).

EXAMPLES

None.

APPLICATION USAGE

As an alternative to setting *errno* to zero before the call and checking if it is non-zero afterwards, applications can use *ferror*( ) to detect whether *psiginfo*( ) or *psignal*( ) encountered an error.

An application wishing to use this method to check for error situations should call *clearerr*(*stderr*) before calling *psiginfo*( ) or *psignal*( ), then if *ferror*(*stderr*) returns non-zero, the value of *errno* indicates which error occurred.

RATIONALE

System V historically has *psignal*( ) and *psiginfo*( ) in **<siginfo.h>**. However, the **<siginfo.h>** header is not specified in the Base Definitions volume of POSIX.1-2024, and the type **siginfo_t** is defined in **<signal.h>**.

FUTURE DIRECTIONS

None.

SEE ALSO

fputc( ), *perror*( ), *strsignal*( )

XBD **<signal.h>**

CHANGE HISTORY

First released in Issue 7.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0447 [399,428], XSH/TC1-2008/0448 [399], and XSH/TC1-2008/0449 [399,401] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0260 [629] is applied.

54804 **NAME**

54805 pthread_atfork — register fork handlers

54806 **SYNOPSIS**

54807 OB
```
#include <pthread.h>

int pthread_atfork(void (*prepare)(void), void (*parent)(void),
    void (*child)(void));
```

54810 **DESCRIPTION**

54811 The *pthread_atfork*( ) function shall declare fork handlers to be called before and after *fork*( ), in
54812 the context of the thread that called *fork*( ). The *prepare* fork handler shall be called before *fork*( )
54813 processing commences. The *parent* fork handle shall be called after *fork*( ) processing completes
54814 in the parent process. The *child* fork handler shall be called after *fork*( ) processing completes in
54815 the child process. If no handling is desired at one or more of these three points, the
54816 corresponding fork handler address(es) may be set to NULL.

54817 If a *fork*( ) call in a multi-threaded process leads to a *child* fork handler calling any function that is
54818 not async-signal-safe, the behavior is undefined.

54819 The order of calls to *pthread_atfork*( ) is significant. The *parent* and *child* fork handlers shall be
54820 called in the order in which they were established by calls to *pthread_atfork*( ). The *prepare* fork
54821 handlers shall be called in the opposite order.

54822 **RETURN VALUE**

54823 Upon successful completion, *pthread_atfork*( ) shall return a value of zero; otherwise, an error
54824 number shall be returned to indicate the error.

54825 **ERRORS**

54826 The *pthread_atfork*( ) function shall fail if:

54827 [ENOMEM] Insufficient table space exists to record the fork handler addresses.

54828 The *pthread_atfork*( ) function shall not return an error code of [EINTR].

54829 **EXAMPLES**

54830 None.

54831 **APPLICATION USAGE**

54832 The original usage pattern envisaged for *pthread_atfork*( ) was for the *prepare* fork handler to lock
54833 mutexes and other locks, and for the *parent* and *child* handlers to unlock them. However, since all
54834 of the relevant unlocking functions, except *sem_post*( ), are not async-signal-safe, this usage
54835 results in undefined behavior in the child process unless the only such unlocking function it calls
54836 is *sem_post*( ).

54837 **RATIONALE**

54838 There are at least two serious problems with the semantics of *fork*( ) in a multi-threaded
54839 program. One problem has to do with state (for example, memory) covered by mutexes.
54840 Consider the case where one thread has a mutex locked and the state covered by that mutex is
54841 inconsistent while another thread calls *fork*( ). In the child, the mutex is in the locked state
54842 (locked by a nonexistent thread and thus can never be unlocked). Having the child simply
54843 reinitialize the mutex is unsatisfactory since this approach does not resolve the question about
54844 how to correct or otherwise deal with the inconsistent state in the child.

54845 It is suggested that programs that use *fork*( ) call an *exec* function very soon afterwards in the
54846 child process, thus resetting all states. In the meantime, only a short list of async-signal-safe
54847 library routines are promised to be available.

Unfortunately, this solution does not address the needs of multi-threaded libraries. Application programs may not be aware that a multi-threaded library is in use, and they feel free to call any number of library routines between the *fork*( ) and *exec* calls, just as they always have. Indeed, they may be extant single-threaded programs and cannot, therefore, be expected to obey new restrictions imposed by the threads library.

On the other hand, the multi-threaded library needs a way to protect its internal state during *fork*( ) in case it is re-entered later in the child process. The problem arises especially in multi-threaded I/O libraries, which are almost sure to be invoked between the *fork*( ) and *exec* calls to effect I/O redirection. The solution may require locking mutex variables during *fork*( ), or it may entail simply resetting the state in the child after the *fork*( ) processing completes.

The *pthread_atfork*( ) function was intended to provide multi-threaded libraries with a means to protect themselves from innocent application programs that call *fork*( ), and to provide multi-threaded application programs with a standard mechanism for protecting themselves from *fork*( ) calls in a library routine or the application itself.

The expected usage was that the prepare handler would acquire all mutex locks and the other two fork handlers would release them.

For example, an application could have supplied a prepare routine that acquires the necessary mutexes the library maintains and supplied child and parent routines that release those mutexes, thus ensuring that the child would have got a consistent snapshot of the state of the library (and that no mutexes would have been left stranded). This is good in theory, but in reality not practical. Each and every mutex and lock in the process must be located and locked. Every component of a program including third-party components must participate and they must agree who is responsible for which mutex or lock. This is especially problematic for mutexes and locks in dynamically allocated memory. All mutexes and locks internal to the implementation must be locked, too. This possibly delays the thread calling *fork*( ) for a long time or even indefinitely since uses of these synchronization objects may not be under control of the application. A final problem to mention here is the problem of locking streams. At least the streams under control of the system (like *stdin*, *stdout*, *stderr*) must be protected by locking the stream with *flockfile*( ). But the application itself could have done that, possibly in the same thread calling *fork*( ). In this case, the process will deadlock.

Alternatively, some libraries might have been able to supply just a *child* routine that reinitializes the mutexes in the library and all associated states to some known value (for example, what it was when the image was originally executed). This approach is not possible, though, because implementations are allowed to fail **_init*( ) and **_destroy*( ) calls for mutexes and locks if the mutex or lock is still locked. In this case, the *child* routine is not able to reinitialize the mutexes and locks.

When *fork*( ) is called, only the calling thread is duplicated in the child process. Synchronization variables remain in the same state in the child as they were in the parent at the time *fork*( ) was called. Thus, for example, mutex locks may be held by threads that no longer exist in the child process, and any associated states may be inconsistent. The intention was that the parent process could have avoided this by explicit code that acquires and releases locks critical to the child via *pthread_atfork*( ). In addition, any critical threads would have needed to be recreated and reinitialized to the proper state in the child (also via *pthread_atfork*( )).

A higher-level package may acquire locks on its own data structures before invoking lower-level packages. Under this scenario, the order specified for fork handler calls allows a simple rule of initialization for avoiding package deadlock: a package initializes all packages on which it depends before it calls the *pthread_atfork*( ) function for itself.

As explained, there is no suitable solution for functionality which requires non-atomic

operations to be protected through mutexes and locks. This is why the POSIX.1 standard since the 1996 release requires that the child process after *fork*( ) in a multi-threaded process only calls async-signal-safe interfaces.

An additional problem arises when *pthread_atfork*( ) is called to register a function in a library that was loaded using *dlopen*( ). If the library is unloaded using *dlclose*( ), and the implementation of *dlclose*( ) does not unregister the function, then when *fork*( ) tries to call it the result will be undefined behavior. Some implementations of *dlclose*( ) do unregister *pthread_atfork*( ) handlers, but this cannot be relied upon by portable applications. The standard provides no portable method for unregistering a function installed as a handler via *pthread_atfork*( ).

FUTURE DIRECTIONS

The *pthread_atfork*( ) function may be removed in a future version of this standard.

SEE ALSO

atexit( ), *exec*, *fork*( )

XBD **<pthread.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 5. Derived from the POSIX Threads Extension.

IEEE PASC Interpretation 1003.1c #4 is applied.

Issue 6

The *pthread_atfork*( ) function is marked as part of the Threads option.

The **<pthread.h>** header is added to the SYNOPSIS.

Issue 7

The *pthread_atfork*( ) function is moved from the Threads option to the Base.

SD5-XSH-ERN-145 is applied, updating the RATIONALE.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0261 [858] is applied.

Issue 8

Austin Group Defect 851 is applied, marking *pthread_atfork*( ) as obsolescent.

54923 **NAME**

54924 pthread_attr_destroy, pthread_attr_init — destroy and initialize the thread attributes object

54925 **SYNOPSIS**

54926 ```
#include <pthread.h>

int pthread_attr_destroy(pthread_attr_t *attr);
int pthread_attr_init(pthread_attr_t *attr);
```

54929 **DESCRIPTION**

54930 The *pthread_attr_destroy*( ) function shall destroy a thread attributes object. An implementation
54931 may cause *pthread_attr_destroy*( ) to set *attr* to an implementation-defined invalid value. A
54932 destroyed *attr* attributes object can be reinitialized using *pthread_attr_init*( ); the results of
54933 otherwise referencing the object after it has been destroyed are undefined.

54934 The *pthread_attr_init*( ) function shall initialize a thread attributes object *attr* with the default
54935 value for all of the individual attributes used by a given implementation.

54936 The resulting attributes object (possibly modified by setting individual attribute values) when
54937 used by *pthread_create*( ) defines the attributes of the thread created. A single attributes object can
54938 be used in multiple simultaneous calls to *pthread_create*( ). Results are undefined if
54939 *pthread_attr_init*( ) is called specifying an already initialized *attr* attributes object.

54940 The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_destroy*( )
54941 does not refer to an initialized thread attributes object.

54942 **RETURN VALUE**

54943 Upon successful completion, *pthread_attr_destroy*( ) and *pthread_attr_init*( ) shall return a value of
54944 0; otherwise, an error number shall be returned to indicate the error.

54945 **ERRORS**

54946 The *pthread_attr_init*( ) function shall fail if:

54947 [ENOMEM] Insufficient memory exists to initialize the thread attributes object.

54948 These functions shall not return an error code of [EINTR].

54949 **EXAMPLES**

54950 None.

54951 **APPLICATION USAGE**

54952 None.

54953 **RATIONALE**

54954 Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to
54955 support probable future standardization in these areas without requiring that the function itself
54956 be changed.

54957 Attributes objects provide clean isolation of the configurable aspects of threads. For example,
54958 ``stack size'' is an important attribute of a thread, but it cannot be expressed portably. When
54959 porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects
54960 can help by allowing the changes to be isolated in a single place, rather than being spread across
54961 every instance of thread creation.

54962 Attributes objects can be used to set up ``classes' of threads with similar attributes; for example,
54963 ``threads with large stacks and high priority'' or ``threads with minimal stacks''. These classes
54964 can be defined in a single place and then referenced wherever threads need to be created.
54965 Changes to ``class'' decisions become straightforward, and detailed analysis of each
54966 *pthread_create*( ) call is not required.

  
```

The attributes objects are defined as opaque types as an aid to extensibility. If these objects had been specified as structures, adding new attributes would force recompilation of all multi-threaded programs when the attributes objects are extended; this might not be possible if different program components were supplied by different vendors.

Additionally, opaque attributes objects present opportunities for improving performance. Argument validity can be checked once when attributes are set, rather than each time a thread is created. Implementations often need to cache kernel objects that are expensive to create. Opaque attributes objects provide an efficient mechanism to detect when cached objects become invalid due to attribute changes.

Since assignment is not necessarily defined on a given opaque type, implementation-defined default values cannot be defined in a portable way. The solution to this problem is to allow attributes objects to be initialized dynamically by attributes object initialization functions, so that default values can be supplied automatically by the implementation.

The following proposal was provided as a suggested alternative to the supplied attributes:

1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to the initialization routines (*pthread_create*( ), *pthread_mutex_init*( ), *pthread_cond_init*( )). The parameter containing the flags should be an opaque type for extensibility. If no flags are set in the parameter, then the objects are created with default characteristics. An implementation may specify implementation-defined flag values and associated behavior.

2. If further specialization of mutexes and condition variables is necessary, implementations may specify additional procedures that operate on the **pthread_mutex_t** and **pthread_cond_t** objects (instead of on attributes objects).

The difficulties with this solution are:

1. A bitmask is not opaque if bits have to be set into bitvector attributes objects using explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**, application programmers need to know the location of each bit. If bits are set or read by encapsulation (that is, get and set functions), then the bitmask is merely an implementation of attributes objects as currently defined and should not be exposed to the programmer.

2. Many attributes are not Boolean or very small integral values. For example, scheduling policy may be placed in 3-bit or 4-bit, but priority requires 5-bit or more, thereby taking up at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this, the bitmask can only reasonably control whether particular attributes are set or not, and it cannot serve as the repository of the value itself. The value needs to be specified as a function parameter (which is non-extensible), or by setting a structure field (which is non-opaque), or by get and set functions (making the bitmask a redundant addition to the attributes objects).

Stack size is defined as an optional attribute because the very notion of a stack is inherently machine-dependent. Some implementations may not be able to change the size of the stack, for example, and others may not need to because stack pages may be discontiguous and can be allocated and released on demand.

The attribute mechanism has been designed in large measure for extensibility. Future extensions to the attribute mechanism or to any attributes object defined in this volume of POSIX.1-2024 has to be done with care so as not to affect binary-compatibility.

Attributes objects, even if allocated by means of dynamic allocation functions such as *malloc*( ),

may have their size fixed at compile time. This means, for example, a *pthread_create*( ) in an implementation with extensions to **pthread_attr_t** cannot look beyond the area that the binary application assumes is valid. This suggests that implementations should maintain a size field in the attributes object, as well as possibly version information, if extensions in different directions (possibly by different vendors) are to be accommodated.

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_destroy*( ) does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_init*( ) refers to an already initialized thread attributes object, it is recommended that the function should fail and report an [EBUSY] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_getstacksize( ), *pthread_attr_getdetachstate*( ), *pthread_create*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_attr_destroy*( ) and *pthread_attr_init*( ) functions are marked as part of the Threads option.

IEEE PASC Interpretation 1003.1 #107 is applied, noting that the effect of initializing an already initialized thread attributes object is undefined.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/71 is applied, updating the ERRORS section to add the optional [EINVAL] error for the *pthread_attr_destroy*( ) function, and the optional [EBUSY] error for the *pthread_attr_init*( ) function.

Issue 7

The *pthread_attr_destroy*( ) and *pthread_attr_init*( ) functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

The [EBUSY] error for an already initialized thread attributes object is removed; this condition results in undefined behavior.

NAME

pthread_attr_getdetachstate, pthread_attr_setdetachstate — get and set the detachstate attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_getdetachstate(const pthread_attr_t *attr,
    int *detachstate);
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
```

DESCRIPTION

The *detachstate* attribute controls whether the thread is created in a detached state. If the thread is created detached, then use of the ID of the newly created thread by the *pthread_detach*( ) or *pthread_join*( ) function is an error.

The *pthread_attr_getdetachstate*( ) and *pthread_attr_setdetachstate*( ) functions, respectively, shall get and set the *detachstate* attribute in the *attr* object.

For *pthread_attr_getdetachstate*( ), *detachstate* shall be set to either PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE.

For *pthread_attr_setdetachstate*( ), the application shall set *detachstate* to either PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE.

A value of PTHREAD_CREATE_DETACHED shall cause all threads created with *attr* to be in the detached state, whereas using a value of PTHREAD_CREATE_JOINABLE shall cause all threads created with *attr* to be in the joinable state. The default value of the *detachstate* attribute shall be PTHREAD_CREATE_JOINABLE.

The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getdetachstate*( ) or *pthread_attr_setdetachstate*( ) does not refer to an initialized thread attributes object.

RETURN VALUE

Upon successful completion, *pthread_attr_getdetachstate*( ) and *pthread_attr_setdetachstate*( ) shall return a value of 0; otherwise, an error number shall be returned to indicate the error.

The *pthread_attr_getdetachstate*( ) function stores the value of the *detachstate* attribute in *detachstate* if successful.

ERRORS

The *pthread_attr_setdetachstate*( ) function shall fail if:

[EINVAL] The value of *detachstate* was not valid

These functions shall not return an error code of [EINTR].

EXAMPLES

Retrieving the detachstate Attribute

This example shows how to obtain the *detachstate* attribute of a thread attribute object.

```
#include <pthread.h>

pthread_attr_t thread_attr;
int            detachstate;
int            rc;

/* code initializing thread_attr */
...
```



```
55088       rc = pthread_attr_getdetachstate (&thread_attr, &detachstate);
55089       if (rc!=0) {
55090           /* handle error */
55091           ...
55092       }
55093       else {
55094           /* legal values for detachstate are:
55095            * PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE
55096            */
55097            ...
55098       }
```

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_getdetachstate*( ) or *pthread_attr_setdetachstate*( ) does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getstacksize*( ), *pthread_create*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_attr_setdetachstate*( ) and *pthread_attr_getdetachstate*( ) functions are marked as part of the Threads option.

The normative text is updated to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/72 is applied, adding the example to the EXAMPLES section.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/73 is applied, updating the ERRORS section to include the optional [EINVAL] error.

Issue 7

The *pthread_attr_setdetachstate*( ) and *pthread_attr_getdetachstate*( ) functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

55125 **NAME**

55126 pthread_attr_getguardsize, pthread_attr_setguardsize — get and set the thread guardsize
55127 attribute

55128 **SYNOPSIS**

55129
```
#include <pthread.h>
```

55130
```
int pthread_attr_getguardsize(const pthread_attr_t *restrict attr,
55131     size_t *restrict guardsize);
55132 int pthread_attr_setguardsize(pthread_attr_t *attr,
55133     size_t guardsize);
```

55134 **DESCRIPTION**

55135 The *pthread_attr_getguardsize*( ) function shall get the *guardsize* attribute in the *attr* object. This
55136 attribute shall be returned in the *guardsize* parameter.

55137 The *pthread_attr_setguardsize*( ) function shall set the *guardsize* attribute in the *attr* object. The new
55138 value of this attribute shall be obtained from the *guardsize* parameter. If *guardsize* is zero, a guard
55139 area shall not be provided for threads created with *attr*. If *guardsize* is greater than zero, a guard
55140 area of at least size *guardsize* bytes shall be provided for each thread created with *attr*.

55141 The *guardsize* attribute controls the size of the guard area for the created thread's stack. The
55142 *guardsize* attribute provides protection against overflow of the stack pointer. If a thread's stack is
55143 created with guard protection, the implementation allocates extra memory at the overflow end
55144 of the stack as a buffer against stack overflow of the stack pointer. If an application overflows
55145 into this buffer an error shall result (possibly in a SIGSEGV signal being delivered to the thread).

55146 A conforming implementation may round up the value contained in *guardsize* to a multiple of
55147 the configurable system variable {PAGESIZE} (see **<sys/mman.h>**). If an implementation
55148 rounds up the value of *guardsize* to a multiple of {PAGESIZE}, a call to *pthread_attr_getguardsize*( )
55149 specifying *attr* shall store in the *guardsize* parameter the guard size specified by the previous
55150 *pthread_attr_setguardsize*( ) function call.

55151 The default value of the *guardsize* attribute is implementation-defined.

55152 If the *stackaddr* attribute has been set (that is, the caller is allocating and managing its own thread
55153 stacks), the *guardsize* attribute shall be ignored and no protection shall be provided by the
55154 implementation. It is the responsibility of the application to manage stack overflow along with
55155 stack allocation and management in this case.

55156 The behavior is undefined if the value specified by the *attr* argument to
55157 *pthread_attr_getguardsize*( ) or *pthread_attr_setguardsize*( ) does not refer to an initialized thread
55158 attributes object.

55159 **RETURN VALUE**

55160 If successful, the *pthread_attr_getguardsize*( ) and *pthread_attr_setguardsize*( ) functions shall return
55161 zero; otherwise, an error number shall be returned to indicate the error.

55162 **ERRORS**

55163 These functions shall fail if:

55164 [EINVAL] The parameter *guardsize* is invalid.

55165 These functions shall not return an error code of [EINTR].

EXAMPLES

Retrieving the guardsize Attribute

This example shows how to obtain the *guardsize* attribute of a thread attribute object.

```
#include <pthread.h>

pthread_attr_t thread_attr;
size_t  guardsize;
int     rc;

/* code initializing thread_attr */
...

rc = pthread_attr_getguardsize (&thread_attr, &guardsize);
if (rc != 0)  {
    /* handle error */
    ...
}
else {
    if (guardsize > 0) {
    /* a guard area of at least guardsize bytes is provided */
    ...
    }
    else {
    /* no guard area provided */
    ...
    }
}
```

APPLICATION USAGE

None.

RATIONALE

The *guardsize* attribute is provided to the application for two reasons:

1. Overflow protection can potentially result in wasted system resources. An application that creates a large number of threads, and which knows its threads never overflow their stack, can save system resources by turning off guard areas.
2. When threads allocate large data structures on the stack, large guard areas may be needed to detect stack overflow.

The default size of the guard area is left implementation-defined since on systems supporting very large page sizes, the overhead might be substantial if at least one guard page is required by default.

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_getguardsize*( ) or *pthread_attr_setguardsize*( ) does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

55207 **SEE ALSO**

55208 XBD **<pthread.h>**, **<sys/mman.h>**

55209 **CHANGE HISTORY**

55210 First released in Issue 5.

55211 **Issue 6**

55212 In the ERRORS section, a third [EINVAL] error condition is removed as it is covered by the
55213 second error condition.

55214 The **restrict** keyword is added to the *pthread_attr_getguardsize*( ) prototype for alignment with the
55215 ISO/IEC 9899: 1999 standard.

55216 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/74 is applied, updating the ERRORS
55217 section to remove the [EINVAL] error (``The attribute *attr* is invalid.''), and replacing it with the
55218 optional [EINVAL] error.

55219 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/76 is applied, adding the example to the
55220 EXAMPLES section.

55221 **Issue 7**

55222 SD5-XSH-ERN-111 is applied, removing the reference to the *stack* attribute in the DESCRIPTION.

55223 SD5-XSH-ERN-175 is applied, updating the DESCRIPTION to note that the default size of the
55224 guard area is implementation-defined.

55225 The *pthread_attr_getguardsize*( ) and *pthread_attr_setguardsize*( ) functions are moved from the XSI
55226 option to the Base.

55227 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition
55228 results in undefined behavior.

NAME

pthread_attr_getinheritsched, pthread_attr_setinheritsched — get and set the inheritsched attribute (**REALTIME THREADS**)

SYNOPSIS

TPS

```
#include <pthread.h>

int pthread_attr_getinheritsched(const pthread_attr_t *restrict attr,
    int *restrict inheritsched);
int pthread_attr_setinheritsched(pthread_attr_t *attr,
    int inheritsched);
```

DESCRIPTION

The *pthread_attr_getinheritsched*( ) and *pthread_attr_setinheritsched*( ) functions, respectively, shall get and set the *inheritsched* attribute in the *attr* argument.

When the attributes objects are used by *pthread_create*( ), the *inheritsched* attribute determines how the other scheduling attributes of the created thread shall be set.

The supported values of *inheritsched* shall be:

PTHREAD_INHERIT_SCHED
: Specifies that the thread scheduling attributes shall be inherited from the creating thread, and the scheduling attributes in this *attr* argument shall be ignored.

PTHREAD_EXPLICIT_SCHED
: Specifies that the thread scheduling attributes shall be set to the corresponding values from this attributes object.

The symbols PTHREAD_INHERIT_SCHED and PTHREAD_EXPLICIT_SCHED are defined in the **<pthread.h>** header.

The following thread scheduling attributes defined by POSIX.1-2024 are affected by the *inheritsched* attribute: scheduling policy (*schedpolicy*), scheduling parameters (*schedparam*), and scheduling contention scope (*contentionscope*).

The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getinheritsched*( ) or *pthread_attr_setinheritsched*( ) does not refer to an initialized thread attributes object.

RETURN VALUE

If successful, the *pthread_attr_getinheritsched*( ) and *pthread_attr_setinheritsched*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_attr_setinheritsched*( ) function shall fail if:

[ENOTSUP] An attempt was made to set the attribute to an unsupported value.

The *pthread_attr_setinheritsched*( ) function may fail if:

[EINVAL] The value of *inheritsched* is not valid.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

After these attributes have been set, a thread can be created with the specified attributes using *pthread_create*( ). Using these routines does not affect the current running thread.

See Section 2.9.4 (on page 540) for further details on thread scheduling attributes and their default settings.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_getinheritsched*( ) or *pthread_attr_setinheritsched*( ) does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getscope*( ), *pthread_attr_getschedpolicy*( ), *pthread_attr_getschedparam*( ), *pthread_create*( )

XBD **<pthread.h>**, **<sched.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

Issue 6

The *pthread_attr_getinheritsched*( ) and *pthread_attr_setinheritsched*( ) functions are marked as part of the Threads and Thread Execution Scheduling options.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Execution Scheduling option.

The **restrict** keyword is added to the *pthread_attr_getinheritsched*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/75 is applied, clarifying the values of *inheritsched* in the DESCRIPTION and adding two optional [EINVAL] errors to the ERRORS section for checking when *attr* refers to an uninitialized thread attribute object.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/77 is applied, adding a reference to Section 2.9.4 (on page 540) in the APPLICATION USAGE section.

Issue 7

The *pthread_attr_getinheritsched*( ) and *pthread_attr_setinheritsched*( ) functions are marked only as part of the Thread Execution Scheduling option as the Threads option is now part of the Base.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0450 [314] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0262 [757] is applied.

55307 **NAME**

55308 pthread_attr_getschedparam, pthread_attr_setschedparam — get and set the schedparam
55309 attribute

55310 **SYNOPSIS**

55311 ```
#include <pthread.h>

int pthread_attr_getschedparam(const pthread_attr_t *restrict attr,
    struct sched_param *restrict param);
int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
    const struct sched_param *restrict param);
```

55316 **DESCRIPTION**

55317 The *pthread_attr_getschedparam*( ) and *pthread_attr_setschedparam*( ) functions, respectively, shall
55318 get and set the scheduling parameter attributes in the *attr* argument. The contents of the *param*
55319 structure are defined in the **<sched.h>** header. For the SCHED_FIFO and SCHED_RR policies,
55320 the only required member of *param* is *sched_priority*.

55321 TSP For the SCHED_SPORADIC policy, the required members of the *param* structure are
55322 *sched_priority*, *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, and
55323 *sched_ss_max_repl*. The specified *sched_ss_repl_period* needs to be greater than or equal to the
55324 specified *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.
55325 The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the
55326 function to succeed; if not, the function shall fail. It is unspecified whether the
55327 *sched_ss_repl_period* and *sched_ss_init_budget* values are stored as provided by this function or are
55328 rounded to align with the resolution of the clock being used.

55329 The behavior is undefined if the value specified by the *attr* argument to
55330 *pthread_attr_getschedparam*( ) or *pthread_attr_setschedparam*( ) does not refer to an initialized thread
55331 attributes object.

55332 **RETURN VALUE**

55333 If successful, the *pthread_attr_getschedparam*( ) and *pthread_attr_setschedparam*( ) functions shall
55334 return zero; otherwise, an error number shall be returned to indicate the error.

55335 **ERRORS**

55336 The *pthread_attr_setschedparam*( ) function shall fail if:

55337 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

55338 The *pthread_attr_setschedparam*( ) function may fail if:

55339 [EINVAL] The value of *param* is not valid.

55340 These functions shall not return an error code of [EINTR].

55341 **EXAMPLES**

55342 None.

55343 **APPLICATION USAGE**

55344 After these attributes have been set, a thread can be created with the specified attributes using
55345 *pthread_create*( ). Using these routines does not affect the current running thread.

55346 **RATIONALE**

55347 If an implementation detects that the value specified by the *attr* argument to
55348 *pthread_attr_getschedparam*( ) or *pthread_attr_setschedparam*( ) does not refer to an initialized thread
55349 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

  
```

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getscope*( ), *pthread_attr_getinheritsched*( ), *pthread_attr_getschedpolicy*( ), *pthread_create*( )

XBD **<pthread.h>**, **<sched.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_attr_getschedparam*( ) and *pthread_attr_setschedparam*( ) functions are marked as part of the Threads option.

The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *pthread_attr_getschedparam*( ) and *pthread_attr_setschedparam*( ) prototypes for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/78 is applied, updating the ERRORS section to include optional errors for the case when *attr* refers to an uninitialized thread attribute object.

Issue 7

The *pthread_attr_getschedparam*( ) and *pthread_attr_setschedparam*( ) functions are moved from the Threads option to the Base.

Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements for the *sched_ss_repl_period* and *sched_ss_init_budget* values.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0451 [314] is applied.

55375 **NAME**

55376 pthread_attr_getschedpolicy, pthread_attr_setschedpolicy — get and set the schedpolicy
55377 attribute (**REALTIME THREADS**)

55378 **SYNOPSIS**

55379 TPS
```
#include <pthread.h>

int pthread_attr_getschedpolicy(const pthread_attr_t *restrict attr,
    int *restrict policy);
int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
```

55383 **DESCRIPTION**

55384 The *pthread_attr_getschedpolicy*( ) and *pthread_attr_setschedpolicy*( ) functions, respectively, shall
55385 get and set the *schedpolicy* attribute in the *attr* argument.

55386 The supported values of *policy* shall include SCHED_FIFO, SCHED_RR, and SCHED_OTHER,
55387 which are defined in the **<sched.h>** header. When threads executing with the scheduling policy
55388 TSP SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC are waiting on a mutex, they shall acquire
55389 the mutex in priority order when the mutex is unlocked.

55390 The behavior is undefined if the value specified by the *attr* argument to
55391 *pthread_attr_getschedpolicy*( ) or *pthread_attr_setschedpolicy*( ) does not refer to an initialized thread
55392 attributes object.

55393 **RETURN VALUE**

55394 If successful, the *pthread_attr_getschedpolicy*( ) and *pthread_attr_setschedpolicy*( ) functions shall
55395 return zero; otherwise, an error number shall be returned to indicate the error.

55396 **ERRORS**

55397 The *pthread_attr_setschedpolicy*( ) function shall fail if:

55398 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

55399 The *pthread_attr_setschedpolicy*( ) function may fail if:

55400 [EINVAL] The value of *policy* is not valid.

55401 These functions shall not return an error code of [EINTR].

55402 **EXAMPLES**

55403 None.

55404 **APPLICATION USAGE**

55405 After these attributes have been set, a thread can be created with the specified attributes using
55406 *pthread_create*( ). Using these routines does not affect the current running thread.

55407 See Section 2.9.4 (on page 540) for further details on thread scheduling attributes and their
55408 default settings.

55409 **RATIONALE**

55410 If an implementation detects that the value specified by the *attr* argument to
55411 *pthread_attr_getschedpolicy*( ) or *pthread_attr_setschedpolicy*( ) does not refer to an initialized thread
55412 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

55413 **FUTURE DIRECTIONS**

55414 None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getscope*( ), *pthread_attr_getinheritsched*( ), *pthread_attr_getschedparam*( ), *pthread_create*( )

XBD **<pthread.h>**, **<sched.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

Issue 6

The *pthread_attr_getschedpolicy*( ) and *pthread_attr_setschedpolicy*( ) functions are marked as part of the Threads and Thread Execution Scheduling options.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Execution Scheduling option.

The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *pthread_attr_getschedpolicy*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/79 is applied, adding a reference to Section 2.9.4 (on page 540) in the APPLICATION USAGE section.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/80 is applied, updating the ERRORS section to include optional errors for the case when *attr* refers to an uninitialized thread attribute object.

Issue 7

The *pthread_attr_getschedpolicy*( ) and *pthread_attr_setschedpolicy*( ) functions are marked only as part of the Thread Execution Scheduling option as the Threads option is now part of the Base.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0452 [314] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0263 [757] is applied.

55442 **NAME**

55443 pthread_attr_getscope, pthread_attr_setscope — get and set the contentionscope attribute
55444 (**REALTIME THREADS**)

55445 **SYNOPSIS**

55446 TPS
```
#include <pthread.h>

int pthread_attr_getscope(const pthread_attr_t *restrict attr,
    int *restrict contentionscope);
int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
```

55450 **DESCRIPTION**

55451 The *pthread_attr_getscope*( ) and *pthread_attr_setscope*( ) functions, respectively, shall get and set
55452 the *contentionscope* attribute in the *attr* object.

55453 The *contentionscope* attribute may have the values PTHREAD_SCOPE_SYSTEM, signifying
55454 system scheduling contention scope, or PTHREAD_SCOPE_PROCESS, signifying process
55455 scheduling contention scope. The symbols PTHREAD_SCOPE_SYSTEM and
55456 PTHREAD_SCOPE_PROCESS are defined in the **<pthread.h>** header.

55457 The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getscope*( ) or
55458 *pthread_attr_setscope*( ) does not refer to an initialized thread attributes object.

55459 **RETURN VALUE**

55460 If successful, the *pthread_attr_getscope*( ) and *pthread_attr_setscope*( ) functions shall return zero;
55461 otherwise, an error number shall be returned to indicate the error.

55462 **ERRORS**

55463 The *pthread_attr_setscope*( ) function shall fail if:

55464 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

55465 The *pthread_attr_setscope*( ) function may fail if:

55466 [EINVAL] The value of *contentionscope* is not valid.

55467 These functions shall not return an error code of [EINTR].

55468 **EXAMPLES**

55469 None.

55470 **APPLICATION USAGE**

55471 After these attributes have been set, a thread can be created with the specified attributes using
55472 *pthread_create*( ). Using these routines does not affect the current running thread.

55473 See Section 2.9.4 (on page 540) for further details on thread scheduling attributes and their
55474 default settings.

55475 **RATIONALE**

55476 If an implementation detects that the value specified by the *attr* argument to
55477 *pthread_attr_getscope*( ) or *pthread_attr_setscope*( ) does not refer to an initialized thread attributes
55478 object, it is recommended that the function should fail and report an [EINVAL] error.

55479 **FUTURE DIRECTIONS**

55480 None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getinheritsched*( ), *pthread_attr_getschedpolicy*( ), *pthread_attr_getschedparam*( ), *pthread_create*( )

XBD **<pthread.h>**, **<sched.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

Issue 6

The *pthread_attr_getscope*( ) and *pthread_attr_setscope*( ) functions are marked as part of the Threads and Thread Execution Scheduling options.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Execution Scheduling option.

The **restrict** keyword is added to the *pthread_attr_getscope*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/81 is applied, adding a reference to Section 2.9.4 (on page 540) in the APPLICATION USAGE section.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/82 is applied, updating the ERRORS section to include optional errors for the case when *attr* refers to an uninitialized thread attribute object.

Issue 7

The *pthread_attr_getscope*( ) and *pthread_attr_setscope*( ) functions are marked only as part of the Thread Execution Scheduling option as the Threads option is now part of the Base.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0453 [314] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0264 [757] is applied.

55507 **NAME**

55508 pthread_attr_getstack, pthread_attr_setstack — get and set stack attributes

55509 **SYNOPSIS**

55510 TSA TSS
```
#include <pthread.h>

int pthread_attr_getstack(const pthread_attr_t *restrict attr,
    void **restrict stackaddr, size_t *restrict stacksize);
int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
    size_t stacksize);
```

55515 **DESCRIPTION**

55516 The *pthread_attr_getstack*( ) and *pthread_attr_setstack*( ) functions, respectively, shall get and set the
55517 thread creation stack attributes *stackaddr* and *stacksize* in the *attr* object.

55518 The stack attributes specify the area of storage to be used for the created thread's stack. The base
55519 (lowest addressable byte) of the storage shall be *stackaddr*, and the size of the storage shall be
55520 *stacksize* bytes. The *stacksize* shall be at least {PTHREAD_STACK_MIN}. The
55521 *pthread_attr_setstack*( ) function may fail with [EINVAL] if *stackaddr* does not meet
55522 implementation-defined alignment requirements. All pages within the stack described by
55523 *stackaddr* and *stacksize* shall be both readable and writable by the thread.

55524 If the *pthread_attr_getstack*( ) function is called before the *stackaddr* attribute has been set, the
55525 behavior is unspecified.

55526 The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getstack*( ) or
55527 *pthread_attr_setstack*( ) does not refer to an initialized thread attributes object.

55528 **RETURN VALUE**

55529 Upon successful completion, these functions shall return a value of 0; otherwise, an error
55530 number shall be returned to indicate the error.

55531 The *pthread_attr_getstack*( ) function shall store the stack attribute values in *stackaddr* and *stacksize*
55532 if successful.

55533 **ERRORS**

55534 The *pthread_attr_setstack*( ) function shall fail if:

55535 [EINVAL] The value of *stacksize* is less than {PTHREAD_STACK_MIN} or exceeds an
55536 implementation-defined limit.

55537 The *pthread_attr_setstack*( ) function may fail if:

55538 [EINVAL] The value of *stackaddr* does not have proper alignment to be used as a stack, or
55539 ((**char** *)*stackaddr* + *stacksize*) lacks proper alignment.

55540 [EACCES] The stack page(s) described by *stackaddr* and *stacksize* are not both readable
55541 and writable by the thread.

55542 These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

These functions are appropriate for use by applications in an environment where the stack for a thread must be placed in some particular region of memory.

While it might seem that an application could detect stack overflow by providing a protected page outside the specified stack region, this cannot be done portably. Implementations are free to place the thread's initial stack pointer anywhere within the specified region to accommodate the machine's stack pointer behavior and allocation requirements. Furthermore, on some architectures, such as the IA-64, ``overflow'' might mean that two separate stack pointers allocated within the region will overlap somewhere in the middle of the region.

After a successful call to *pthread_attr_setstack*( ), the storage area specified by the *stackaddr* parameter is under the control of the implementation, as described in Section 2.9.8 (on page 548).

The specification of the *stackaddr* attribute presents several ambiguities that make portable use of these functions impossible. For example, the standard allows implementations to impose arbitrary alignment requirements on *stackaddr*. Applications cannot assume that a buffer obtained from *malloc*( ) is suitably aligned. Note that although the *stacksize* value passed to *pthread_attr_setstack*( ) must satisfy alignment requirements, the same is not true for *pthread_attr_setstacksize*( ) where the implementation must increase the specified size if necessary to achieve the proper alignment.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_getstack*( ) or *pthread_attr_setstack*( ) does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getdetachstate*( ), *pthread_attr_getstacksize*( ), *pthread_create*( )

XBD **<limits.h>**, **<pthread.h>**

CHANGE HISTORY

First released in Issue 6. Developed as part of the XSI option and brought into the BASE by IEEE PASC Interpretation 1003.1 #101.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/83 is applied, updating the APPLICATION USAGE section to refer to Section 2.9.8 (on page 548).

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC/D6/84 is applied, updating the ERRORS section to include optional errors for the case when *attr* refers to an uninitialized thread attribute object.

Issue 7

SD5-XSH-ERN-66 is applied, correcting the use of *attr* in the [EINVAL] error condition.

Austin Group Interpretation 1003.1-2001 #057 is applied, clarifying the behavior if the function is called before the *stackaddr* attribute is set.

SD5-XSH-ERN-157 is applied, updating the APPLICATION USAGE section.

The description of the *stackaddr* attribute is updated in the DESCRIPTION and APPLICATION USAGE sections.

55587 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition
55588 results in undefined behavior.

NAME

pthread_attr_getstacksize, pthread_attr_setstacksize — get and set the stacksize attribute

SYNOPSIS

TSS

```
#include <pthread.h>

int pthread_attr_getstacksize(const pthread_attr_t *restrict attr,
    size_t *restrict stacksize);
int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

DESCRIPTION

The *pthread_attr_getstacksize*( ) and *pthread_attr_setstacksize*( ) functions, respectively, shall get and set the thread creation *stacksize* attribute in the *attr* object.

The *stacksize* attribute shall define the minimum stack size (in bytes) allocated for the created threads stack.

The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getstacksize*( ) or *pthread_attr_setstacksize*( ) does not refer to an initialized thread attributes object.

RETURN VALUE

Upon successful completion, *pthread_attr_getstacksize*( ) and *pthread_attr_setstacksize*( ) shall return a value of 0; otherwise, an error number shall be returned to indicate the error.

The *pthread_attr_getstacksize*( ) function stores the *stacksize* attribute value in *stacksize* if successful.

ERRORS

The *pthread_attr_setstacksize*( ) function shall fail if:

[EINVAL] The value of *stacksize* is less than {PTHREAD_STACK_MIN} or exceeds a system-imposed limit.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_getstacksize*( ) or *pthread_attr_setstacksize*( ) does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getdetachstate*( ), *pthread_create*( )

XBD **<limits.h>**, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_attr_getstacksize*( ) and *pthread_attr_setstacksize*( ) functions are marked as part of the Threads and Thread Stack Size Attribute options.

The **restrict** keyword is added to the *pthread_attr_getstacksize*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/43 is applied, correcting the margin code in the SYNOPSIS from TSA to TSS and updating the CHANGE HISTORY from ``Thread Stack Address Attribute'' option to ``Thread Stack Size Attribute'' option.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/87 is applied, updating the ERRORS section to include optional errors for the case when *attr* refers to an uninitialized thread attribute object.

Issue 7

The *pthread_attr_getstacksize*( ) and *pthread_attr_setstacksize*( ) functions are marked only as part of the Thread Stack Size Attribute option as the Threads option is now part of the Base.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0265 [757] is applied.

55646 **NAME**

55647 pthread_attr_init — initialize the thread attributes object

55648 **SYNOPSIS**

55649 `#include <pthread.h>`

55650 `int pthread_attr_init(pthread_attr_t *attr);`

55651 **DESCRIPTION**

55652 Refer to *pthread_attr_destroy( )*.

55653 **NAME**

55654 pthread_attr_setdetachstate — set the detachstate attribute

55655 **SYNOPSIS**

55656 `#include <pthread.h>`

55657 `int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);`

55658 **DESCRIPTION**

55659 Refer to *pthread_attr_getdetachstate( )*.

55660 **NAME**

55661 pthread_attr_setguardsize — set the thread guardsize attribute

55662 **SYNOPSIS**

55663 ```
#include <pthread.h>
```

55664 ```
int pthread_attr_setguardsize(pthread_attr_t *attr,
```

55665 ```
    size_t guardsize);
```

55666 **DESCRIPTION**

55667 Refer to *pthread_attr_getguardsize( )*.

  
```

NAME

pthread_attr_setinheritsched — set the inheritsched attribute (**REALTIME THREADS**)

SYNOPSIS

TPS

```
#include <pthread.h>

int pthread_attr_setinheritsched(pthread_attr_t *attr,
    int inheritsched);
```

DESCRIPTION

Refer to *pthread_attr_getinheritsched( )*.

55676 **NAME**

55677 pthread_attr_setschedparam — set the schedparam attribute

55678 **SYNOPSIS**

55679 ```
#include <pthread.h>
```

55680 ```
int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
```

55681 ```
    const struct sched_param *restrict param);
```

55682 **DESCRIPTION**

55683 Refer to *pthread_attr_getschedparam( )*.

  
```

NAME

pthread_attr_setschedpolicy — set the schedpolicy attribute (**REALTIME THREADS**)

SYNOPSIS

TPS

```
#include <pthread.h>

int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
```

DESCRIPTION

Refer to *pthread_attr_getschedpolicy( )*.

55691 **NAME**

55692 pthread_attr_setscope — set the contentionscope attribute (**REALTIME THREADS**)

55693 **SYNOPSIS**

55694 TPS
```
#include <pthread.h>

int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
```

55695

55696 **DESCRIPTION**

55697 Refer to *pthread_attr_getscope( )*.

NAME

pthread_attr_setstack — set the stack attribute

SYNOPSIS

TSA TSS

```
#include <pthread.h>

int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
    size_t stacksize);
```

DESCRIPTION

Refer to *pthread_attr_getstack( )*.

NAME

pthread_attr_setstacksize — set the stacksize attribute

SYNOPSIS

TSS

```
#include <pthread.h>

int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

DESCRIPTION

Refer to *pthread_attr_getstacksize( )*.

55713 **NAME**

55714 pthread_barrier_destroy, pthread_barrier_init — destroy and initialize a barrier object

55715 **SYNOPSIS**

55716 ```
#include <pthread.h>

int pthread_barrier_destroy(pthread_barrier_t *barrier);
int pthread_barrier_init(pthread_barrier_t *restrict barrier,
    const pthread_barrierattr_t *restrict attr, unsigned count);
```

55720 **DESCRIPTION**

55721 The *pthread_barrier_destroy*( ) function shall destroy the barrier referenced by *barrier* and release
55722 any resources used by the barrier. The effect of subsequent use of the barrier is undefined until
55723 the barrier is reinitialized by another call to *pthread_barrier_init*( ). An implementation may use
55724 this function to set *barrier* to an invalid value. The results are undefined if
55725 *pthread_barrier_destroy*( ) is called when any thread is blocked on the barrier, or if this function is
55726 called with an uninitialized barrier.

55727 The *pthread_barrier_init*( ) function shall allocate any resources required to use the barrier
55728 referenced by *barrier* and shall initialize the barrier with attributes referenced by *attr*. If *attr* is
55729 NULL, the default barrier attributes shall be used; the effect is the same as passing the address of
55730 a default barrier attributes object. The results are undefined if *pthread_barrier_init*( ) is called
55731 when any thread is blocked on the barrier (that is, has not returned from the
55732 *pthread_barrier_wait*( ) call). The results are undefined if a barrier is used without first being
55733 initialized. The results are undefined if *pthread_barrier_init*( ) is called specifying an already
55734 initialized barrier.

55735 The *count* argument specifies the number of threads that have to call *pthread_barrier_wait*( )
55736 before any of them successfully return from the call. The value specified by *count* needs to be
55737 greater than zero.

55738 If the *pthread_barrier_init*( ) function fails, the barrier shall not be initialized and the contents of
55739 *barrier* are undefined.

55740 See Section 2.9.9 (on page 548) for further requirements.

55741 **RETURN VALUE**

55742 Upon successful completion, these functions shall return zero; otherwise, an error number shall
55743 be returned to indicate the error.

55744 **ERRORS**

55745 The *pthread_barrier_init*( ) function shall fail if:

55746 [EAGAIN] The system lacks the necessary resources to initialize another barrier.

55747 [EINVAL] The value specified by *count* is equal to zero.

55748 [ENOMEM] Insufficient memory exists to initialize the barrier.

55749 These functions shall not return an error code of [EINTR].

  
```

55750 **EXAMPLES**

55751 None.

55752 **APPLICATION USAGE**

55753 None.

55754 **RATIONALE**

55755 If an implementation detects that the value specified by the *barrier* argument to
55756 *pthread_barrier_destroy*( ) does not refer to an initialized barrier object, it is recommended that the
55757 function should fail and report an [EINVAL] error.

55758 If an implementation detects that the value specified by the *attr* argument to
55759 *pthread_barrier_init*( ) does not refer to an initialized barrier attributes object, it is recommended
55760 that the function should fail and report an [EINVAL] error.

55761 If an implementation detects that the value specified by the *barrier* argument to
55762 *pthread_barrier_destroy*( ) or *pthread_barrier_init*( ) refers to a barrier that is in use (for example, in
55763 a *pthread_barrier_wait*( ) call) by another thread, or detects that the value specified by the *barrier*
55764 argument to *pthread_barrier_init*( ) refers to an already initialized barrier object, it is
55765 recommended that the function should fail and report an [EBUSY] error.

55766 **FUTURE DIRECTIONS**

55767 None.

55768 **SEE ALSO**

55769 *pthread_barrier_wait*( )

55770 XBD **<pthread.h>**

55771 **CHANGE HISTORY**

55772 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

55773 **Issue 7**

55774 The *pthread_barrier_destroy*( ) and *pthread_barrier_init*( ) functions are moved from the Barriers
55775 option to the Base.

55776 The [EINVAL] error for an uninitialized barrier object and an uninitialized barrier attributes
55777 object is removed; this condition results in undefined behavior.

55778 The [EBUSY] error for a barrier that is in use or an already initialized barrier object is removed;
55779 this condition results in undefined behavior.

55780 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0266 [972] is applied.

NAME

pthread_barrier_wait — synchronize at a barrier

SYNOPSIS

```
#include <pthread.h>

int pthread_barrier_wait(pthread_barrier_t *barrier);
```

DESCRIPTION

The *pthread_barrier_wait*( ) function shall synchronize participating threads at the barrier referenced by *barrier*. The calling thread shall block until the required number of threads have called *pthread_barrier_wait*( ) specifying the barrier.

When the required number of threads have called *pthread_barrier_wait*( ) specifying the barrier, the constant PTHREAD_BARRIER_SERIAL_THREAD shall be returned to one unspecified thread and zero shall be returned to each of the remaining threads. At this point, the barrier shall be reset to the state it had as a result of the most recent *pthread_barrier_init*( ) function that referenced it.

The constant PTHREAD_BARRIER_SERIAL_THREAD is defined in **<pthread.h>** and its value shall be distinct from any other value returned by *pthread_barrier_wait*( ).

The results are undefined if this function is called with an uninitialized barrier.

If a signal is delivered to a thread blocked on a barrier, upon return from the signal handler the thread shall resume waiting at the barrier if the barrier wait has not completed (that is, if the required number of threads have not arrived at the barrier during the execution of the signal handler); otherwise, the thread shall continue as normal from the completed barrier wait. Until the thread in the signal handler returns from it, it is unspecified whether other threads may proceed past the barrier once they have all reached it.

A thread that has blocked on a barrier shall not prevent any unblocked thread that is eligible to use the same processing resources from eventually making forward progress in its execution. Eligibility for processing resources shall be determined by the scheduling policy.

RETURN VALUE

Upon successful completion, the *pthread_barrier_wait*( ) function shall return PTHREAD_BARRIER_SERIAL_THREAD for a single (arbitrary) thread synchronized at the barrier and zero for each of the other threads. Otherwise, an error number shall be returned to indicate the error.

ERRORS

This function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications using this function may be subject to priority inversion, as discussed in XBD Section 3.275 (on page 72).

RATIONALE

If an implementation detects that the value specified by the *barrier* argument to *pthread_barrier_wait*( ) does not refer to an initialized barrier object, it is recommended that the function should fail and report an [EINVAL] error.

55823 **FUTURE DIRECTIONS**

55824 None.

55825 **SEE ALSO**

55826 *pthread_barrier_destroy*( )

55827 XBD Section 3.275 (on page 72), Section 4.15.2 (on page 104), **<pthread.h>**

55828 **CHANGE HISTORY**

55829 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

55830 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

55831 **Issue 7**

55832 The *pthread_barrier_wait*( ) function is moved from the Barriers option to the Base.

55833 The [EINVAL] error for an uninitialized barrier object is removed; this condition results in
55834 undefined behavior.

NAME

pthread_barrierattr_destroy, pthread_barrierattr_init — destroy and initialize the barrier attributes object

SYNOPSIS

```
#include <pthread.h>

int pthread_barrierattr_destroy(pthread_barrierattr_t *attr);
int pthread_barrierattr_init(pthread_barrierattr_t *attr);
```

DESCRIPTION

The *pthread_barrierattr_destroy*( ) function shall destroy a barrier attributes object. A destroyed *attr* attributes object can be reinitialized using *pthread_barrierattr_init*( ); the results of otherwise referencing the object after it has been destroyed are undefined. An implementation may cause *pthread_barrierattr_destroy*( ) to set the object referenced by *attr* to an invalid value.

The *pthread_barrierattr_init*( ) function shall initialize a barrier attributes object *attr* with the default value for all of the attributes defined by the implementation.

If *pthread_barrierattr_init*( ) is called specifying an already initialized *attr* attributes object, the results are undefined.

After a barrier attributes object has been used to initialize one or more barriers, any function affecting the attributes object (including destruction) shall not affect any previously initialized barrier.

The behavior is undefined if the value specified by the *attr* argument to *pthread_barrierattr_destroy*( ) does not refer to an initialized barrier attributes object.

RETURN VALUE

If successful, the *pthread_barrierattr_destroy*( ) and *pthread_barrierattr_init*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_barrierattr_init*( ) function shall fail if:

[ENOMEM] Insufficient memory exists to initialize the barrier attributes object.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_barrierattr_destroy*( ) does not refer to an initialized barrier attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_barrierattr_getpshared( )

XBD **<pthread.h>**

55876 **CHANGE HISTORY**

55877 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

55878 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

55879 **Issue 7**

55880 The *pthread_barrierattr_destroy*( ) and *pthread_barrierattr_init*( ) functions are moved from the
55881 Barriers option to the Base.

55882 The [EINVAL] error for an uninitialized barrier attributes object is removed; this condition
55883 results in undefined behavior.

NAME

pthread_barrierattr_getpshared, pthread_barrierattr_setpshared — get and set the process-shared attribute of the barrier attributes object

SYNOPSIS

TSH
```
#include <pthread.h>

int pthread_barrierattr_getpshared(const pthread_barrierattr_t
    *restrict attr, int *restrict pshared);
int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
    int pshared);
```

DESCRIPTION

The *pthread_barrierattr_getpshared*( ) function shall obtain the value of the *process-shared* attribute from the attributes object referenced by *attr*. The *pthread_barrierattr_setpshared*( ) function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

The *process-shared* attribute is set to PTHREAD_PROCESS_SHARED to permit a barrier to be operated upon by any thread that has access to the memory where the barrier is allocated. See Section 2.9.9 (on page 548) for further requirements. The default value of the attribute shall be PTHREAD_PROCESS_PRIVATE. Both constants PTHREAD_PROCESS_SHARED and PTHREAD_PROCESS_PRIVATE are defined in **<pthread.h>**.

Additional attributes, their default values, and the names of the associated functions to get and set those attribute values are implementation-defined.

The behavior is undefined if the value specified by the *attr* argument to *pthread_barrierattr_getpshared*( ) or *pthread_barrierattr_setpshared*( ) does not refer to an initialized barrier attributes object.

RETURN VALUE

If successful, the *pthread_barrierattr_getpshared*( ) function shall return zero and store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise, an error number shall be returned to indicate the error.

If successful, the *pthread_barrierattr_setpshared*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_barrierattr_setpshared*( ) function may fail if:

[EINVAL] The new value specified for the *process-shared* attribute is not one of the legal values PTHREAD_PROCESS_SHARED or PTHREAD_PROCESS_PRIVATE.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

The *pthread_barrierattr_getpshared*( ) and *pthread_barrierattr_setpshared*( ) functions are part of the Thread Process-Shared Synchronization option and need not be provided on all implementations.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_barrierattr_getpshared*( ) or *pthread_barrierattr_setpshared*( ) does not refer to an initialized barrier attributes object, it is recommended that the function should fail and report an [EINVAL]

55928 error.

55929 **FUTURE DIRECTIONS**

55930 None.

55931 **SEE ALSO**

55932 *pthread_barrier_destroy*( ), *pthread_barrierattr_destroy*( )

55933 XBD **<pthread.h>**

55934 **CHANGE HISTORY**

55935 First released in Issue 6. Derived from IEEE Std 1003.1j-2000

55936 **Issue 7**

55937 The *pthread_barrierattr_getpshared*( ) and *pthread_barrierattr_setpshared*( ) functions are marked
55938 only as part of the Thread Process-Shared Synchronization option as the Threads option is now
55939 part of the Base.

55940 The [EINVAL] error for an uninitialized barrier attributes object is removed; this condition
55941 results in undefined behavior.

55942 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0266 [972] and XSH/TC2-2008/0267
55943 [757] are applied.

55944 **NAME**

55945 pthread_barrierattr_init — initialize the barrier attributes object

55946 **SYNOPSIS**

55947 ```
#include <pthread.h>
```

55948 ```
int pthread_barrierattr_init(pthread_barrierattr_t *attr);
```

55949 **DESCRIPTION**

55950 Refer to *pthread_barrierattr_destroy*( ).

55951 **NAME**

55952 pthread_barrierattr_setpshared — set the process-shared attribute of the barrier attributes object

55953 **SYNOPSIS**

55954 TSH
```
#include <pthread.h>

int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
    int pshared);
```

55957 **DESCRIPTION**

55958 Refer to *pthread_barrierattr_getpshared( )*.

55959 **NAME**

55960 pthread_cancel — cancel execution of a thread

55961 **SYNOPSIS**

55962 ```
#include <pthread.h>
```

55963 ```
int pthread_cancel(pthread_t thread);
```

55964 **DESCRIPTION**

55965 The *pthread_cancel*( ) function shall request that *thread* be canceled. The target thread's
55966 cancelability state and type determines when the cancellation takes effect. When the cancellation
55967 is acted on, the cancellation cleanup handlers for *thread* shall be called. When the last
55968 cancellation cleanup handler returns, the thread-specific data destructor functions shall be called
55969 for *thread*. When the last destructor function returns, *thread* shall be terminated. It shall not be an
55970 error to request cancellation of a zombie thread.

55971 The cancellation processing in the target thread shall run asynchronously with respect to the
55972 calling thread returning from *pthread_cancel*( ).

55973 If *thread* refers to a thread that was created using *thrd_create*( ), the behavior is undefined.

55974 **RETURN VALUE**

55975 If successful, the *pthread_cancel*( ) function shall return zero; otherwise, an error number shall be
55976 returned to indicate the error.

55977 **ERRORS**

55978 The *pthread_cancel*( ) function shall not return an error code of [EINTR].

55979 **EXAMPLES**

55980 None.

55981 **APPLICATION USAGE**

55982 None.

55983 **RATIONALE**

55984 Two alternative functions were considered for sending the cancellation notification to a thread.
55985 One would be to define a new SIGCANCEL signal that had the cancellation semantics when
55986 delivered; the other was to define the new *pthread_cancel*( ) function, which would trigger the
55987 cancellation semantics.

55988 The advantage of a new signal was that so much of the delivery criteria were identical to that
55989 used when trying to deliver a signal that making cancellation notification a signal was seen as
55990 consistent. Indeed, many implementations implement cancellation using a special signal. On the
55991 other hand, there would be no signal functions that could be used with this signal except
55992 *pthread_kill*( ), and the behavior of the delivered cancellation signal would be unlike any
55993 previously existing defined signal.

55994 The benefits of a special function include the recognition that this signal would be defined
55995 because of the similar delivery criteria and that this is the only common behavior between a
55996 cancellation request and a signal. In addition, the cancellation delivery mechanism does not
55997 have to be implemented as a signal. There are also strong, if not stronger, parallels with
55998 language exception mechanisms than with signals that are potentially obscured if the delivery
55999 mechanism is visibly closer to signals.

56000 In the end, it was considered that as there were so many exceptions to the use of the new signal
56001 with existing signals functions it would be misleading. A special function has resolved this
56002 problem. This function was carefully defined so that an implementation wishing to provide the
56003 cancellation functions on top of signals could do so. The special function also means that
56004 implementations are not obliged to implement cancellation with signals.

If an implementation detects use of a thread ID after the end of its lifetime, it is recommended that the function should fail and report an [ESRCH] error.

Historical implementations varied on the result of a *pthread_cancel*( ) with a thread ID indicating a zombie thread. Some indicated success with nothing further to do because the thread had already terminated, while others gave an error of [ESRCH]. Since the definition of thread lifetime in this standard covers zombie threads, the [ESRCH] error as described is inappropriate in this case and implementations that give this error do not conform.

Use of *pthread_cancel*( ) to cancel a thread that was created using *thrd_create*( ) is undefined because *thrd_join*( ) has no way to indicate a thread was cancelled. The standard developers considered adding a `thrd_canceled` enumeration constant that *thrd_join*( ) would return in this case. However, this return would be unexpected in code that is written to conform to the ISO C standard, and it would also not solve the problem that threads which use only ISO C **<threads.h>** interfaces (such as ones created by third party libraries written to conform to the ISO C standard) have no way to handle being cancelled, as the ISO C standard does not provide cancellation cleanup handlers.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_exit( ), *pthread_cond_clockwait*( ), *pthread_join*( ), *pthread_setcancelstate*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_cancel*( ) function is marked as part of the Threads option.

Issue 7

The *pthread_cancel*( ) function is moved from the Threads option to the Base.

Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

Issue 8

Austin Group Defect 792 is applied, adding a requirement that passing the thread ID of a zombie thread to *pthread_cancel*( ) is not treated as an error.

Austin Group Defect 1302 is applied, updating the page to account for the addition of **<threads.h>** interfaces.

56037 **NAME**

56038 pthread_cleanup_pop, pthread_cleanup_push — establish cancellation handlers

56039 **SYNOPSIS**

56040
```
#include <pthread.h>
```

56041
```
void pthread_cleanup_pop(int execute);
```
56042
```
void pthread_cleanup_push(void (*routine)(void*), void *arg);
```

56043 **DESCRIPTION**

56044 The *pthread_cleanup_pop*( ) function shall remove the routine at the top of the calling thread's
56045 cancellation cleanup stack and optionally invoke it (if *execute* is non-zero).

56046 The *pthread_cleanup_push*( ) function shall push the specified cancellation cleanup handler *routine*
56047 onto the calling thread's cancellation cleanup stack. The cancellation cleanup handler shall be
56048 popped from the cancellation cleanup stack and invoked with the argument *arg* when:

56049 • The thread exits (that is, calls *pthread_exit*( )).

56050 • The thread acts upon a cancellation request.

56051 • The thread calls *pthread_cleanup_pop*( ) with a non-zero *execute* argument.

56052 It is unspecified whether *pthread_cleanup_push*( ) and *pthread_cleanup_pop*( ) are macros or
56053 functions. If a macro definition is suppressed in order to access an actual function, or a program
56054 defines an external identifier with any of these names, the behavior is undefined. The
56055 application shall ensure that they appear as statements, and in pairs within the same lexical
56056 scope (that is, the *pthread_cleanup_push*( ) macro may be thought to expand to a token list whose
56057 first token is `'{'` with *pthread_cleanup_pop*( ) expanding to a token list whose last token is the
56058 corresponding `'}'`).

56059 The effect of calling *longjmp*( ) or *siglongjmp*( ) is undefined if there have been any calls to
56060 *pthread_cleanup_push*( ) or *pthread_cleanup_pop*( ) made without the matching call since the jump
56061 buffer was filled. The effect of calling *longjmp*( ) or *siglongjmp*( ) from inside a cancellation
56062 cleanup handler is also undefined unless the jump buffer was also filled in the cancellation
56063 cleanup handler.

56064 Invoking a cancellation cleanup handler may terminate the execution of any code block being
56065 executed by the thread whose execution began after the corresponding invocation of
56066 *pthread_cleanup_push*( ).

56067 The effect of the use of **return**, **break**, **continue**, and **goto** to prematurely leave a code block
56068 described by a pair of *pthread_cleanup_push*( ) and *pthread_cleanup_pop*( ) functions calls is
56069 undefined.

56070 **RETURN VALUE**

56071 The *pthread_cleanup_push*( ) and *pthread_cleanup_pop*( ) functions shall not return a value.

56072 **ERRORS**

56073 No errors are defined.

56074 These functions shall not return an error code of [EINTR].

EXAMPLES

The following is an example using thread primitives to implement a cancelable, writers-priority read-write lock:

```
typedef struct {
    pthread_mutex_t lock;
    pthread_cond_t rcond,
        wcond;
    int lock_count; /* < 0 .. Held by writer. */
                    /* > 0 .. Held by lock_count readers. */
                    /* = 0 .. Held by nobody. */
    int waiting_writers; /* Count of waiting writers. */
} rwlock;

void
waiting_reader_cleanup(void *arg)
{
    rwlock *l;

    l = (rwlock *) arg;
    pthread_mutex_unlock(&l->lock);
}

void
lock_for_read(rwlock *l)
{
    pthread_mutex_lock(&l->lock);
    pthread_cleanup_push(waiting_reader_cleanup, l);
    while ((l->lock_count < 0) || (l->waiting_writers != 0))
        pthread_cond_wait(&l->rcond, &l->lock);
    l->lock_count++;
   /*
    * Note the pthread_cleanup_pop executes
    * waiting_reader_cleanup.
    */
    pthread_cleanup_pop(1);
}

void
release_read_lock(rwlock *l)
{
    pthread_mutex_lock(&l->lock);
    if (--l->lock_count == 0)
        pthread_cond_signal(&l->wcond);
    pthread_mutex_unlock(&l->lock);
}

void
waiting_writer_cleanup(void *arg)
{
    rwlock *l;

    l = (rwlock *) arg;
    if ((--l->waiting_writers == 0) && (l->lock_count >= 0)) {
       /*
```



```
56123                 * This only happens if we have been canceled. If the
56124                 * lock is not held by a writer, there may be readers who
56125                 * were blocked because waiting_writers was positive; they
56126                 * can now be unblocked.
56127                 */
56128                pthread_cond_broadcast(&l->rcond);
56129            }
56130            pthread_mutex_unlock(&l->lock);
56131        }
56132        void
56133        lock_for_write(rwlock *l)
56134        {
56135            pthread_mutex_lock(&l->lock);
56136            l->waiting_writers++;
56137            pthread_cleanup_push(waiting_writer_cleanup, l);
56138            while (l->lock_count != 0)
56139                pthread_cond_wait(&l->wcond, &l->lock);
56140            l->lock_count = -1;
56141           /*
56142            * Note the pthread_cleanup_pop executes
56143            * waiting_writer_cleanup.
56144            */
56145            pthread_cleanup_pop(1);
56146        }
56147        void
56148        release_write_lock(rwlock *l)
56149        {
56150            pthread_mutex_lock(&l->lock);
56151            l->lock_count = 0;
56152            if (l->waiting_writers == 0)
56153                pthread_cond_broadcast(&l->rcond);
56154            else
56155                pthread_cond_signal(&l->wcond);
56156            pthread_mutex_unlock(&l->lock);
56157        }
56158        /*
56159         * This function is called to initialize the read/write lock.
56160         */
56161        void
56162        initialize_rwlock(rwlock *l)
56163        {
56164            pthread_mutex_init(&l->lock, pthread_mutexattr_default);
56165            pthread_cond_init(&l->wcond, pthread_condattr_default);
56166            pthread_cond_init(&l->rcond, pthread_condattr_default);
56167            l->lock_count = 0;
56168            l->waiting_writers = 0;
56169        }
56170        reader_thread()
56171        {
56172            lock_for_read(&lock);
```

```
        pthread_cleanup_push(release_read_lock, &lock);
       /*
        * Thread has read lock.
        */
        pthread_cleanup_pop(1);
    }

    writer_thread()
    {
        lock_for_write(&lock);
        pthread_cleanup_push(release_write_lock, &lock);
       /*
        * Thread has write lock.
        */
    pthread_cleanup_pop(1);
    }
```

APPLICATION USAGE

The two routines that push and pop cancellation cleanup handlers, *pthread_cleanup_push*( ) and *pthread_cleanup_pop*( ), can be thought of as left and right-parentheses. They always need to be matched.

RATIONALE

The restriction that the two routines that push and pop cancellation cleanup handlers, *pthread_cleanup_push*( ) and *pthread_cleanup_pop*( ), have to appear in the same lexical scope allows for efficient macro or compiler implementations and efficient storage management. A sample implementation of these routines as macros might look like this:

```
#define pthread_cleanup_push(rtn,arg) { \
    struct _pthread_handler_rec __cleanup_handler, **__head; \
    __cleanup_handler.rtn = rtn; \
    __cleanup_handler.arg = arg; \
    (void) pthread_getspecific(_pthread_handler_key, &__head); \
    __cleanup_handler.next = *__head; \
    *__head = &__cleanup_handler;

#define pthread_cleanup_pop(ex) \
    *__head = __cleanup_handler.next; \
    if (ex) (*__cleanup_handler.rtn)(__cleanup_handler.arg); \
}
```

A more ambitious implementation of these routines might do even better by allowing the compiler to note that the cancellation cleanup handler is a constant and can be expanded inline.

This volume of POSIX.1-2024 currently leaves unspecified the effect of calling *longjmp*( ) from a signal handler executing in a POSIX System Interfaces function. If an implementation wants to allow this and give the programmer reasonable behavior, the *longjmp*( ) function has to call all cancellation cleanup handlers that have been pushed but not popped since the time *setjmp*( ) was called.

Consider a multi-threaded function called by a thread that uses signals. If a signal were delivered to a signal handler during the operation of *qsort*( ) and that handler were to call *longjmp*( ) (which, in turn, did *not* call the cancellation cleanup handlers) the helper threads created by the *qsort*( ) function would not be canceled. Instead, they would continue to execute and write into the argument array even though the array might have been popped off the stack.

Note that the specified cleanup handling mechanism is especially tied to the C language and, while the requirement for a uniform mechanism for expressing cleanup is language-independent, the mechanism used in other languages may be quite different. In addition, this mechanism is really only necessary due to the lack of a real exception mechanism in the C language, which would be the ideal solution.

There is no notion of a cancellation cleanup-safe function. If an application has no cancellation points in its signal handlers, blocks any signal whose handler may have cancellation points while calling async-unsafe functions, or disables cancellation while calling async-unsafe functions, all functions may be safely called from cancellation cleanup routines.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cancel( ), *pthread_setcancelstate*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_cleanup_pop*( ) and *pthread_cleanup_push*( ) functions are marked as part of the Threads option.

The APPLICATION USAGE section is added.

The normative text is updated to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/88 is applied, updating the DESCRIPTION to describe the consequences of prematurely leaving a code block defined by the *pthread_cleanup_push*( ) and *pthread_cleanup_pop*( ) functions.

Issue 7

The *pthread_cleanup_pop*( ) and *pthread_cleanup_push*( ) functions are moved from the Threads option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0454 [229] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0268 [624] is applied.

Issue 8

Austin Group Defect 613 is applied, clarifying the relationship of automatic object lifetimes to cancellation cleanup functions.

NAME

pthread_cond_broadcast, pthread_cond_signal — broadcast or signal a condition

SYNOPSIS

```
#include <pthread.h>

int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_signal(pthread_cond_t *cond);
```

DESCRIPTION

These functions shall unblock threads blocked on a condition variable.

The *pthread_cond_broadcast*( ) function shall, as a single atomic operation, determine which threads, if any, are blocked on the specified condition variable *cond* and unblock all of these threads.

The *pthread_cond_signal*( ) function shall, as a single atomic operation, determine which threads, if any, are blocked on the specified condition variable *cond* and unblock at least one of these threads.

If more than one thread is blocked on a condition variable, the scheduling policy shall determine the order in which threads are unblocked. When each thread unblocked as a result of a *pthread_cond_broadcast*( ) or *pthread_cond_signal*( ) returns from its call to *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), or *pthread_cond_wait*( ), the thread shall own the mutex with which it called *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), or *pthread_cond_wait*( ). The thread(s) that are unblocked shall contend for the mutex according to the scheduling policy (if applicable), and as if each had called *pthread_mutex_lock*( ).

The *pthread_cond_broadcast*( ) or *pthread_cond_signal*( ) functions may be called by a thread whether or not it currently owns the mutex that threads calling *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), or *pthread_cond_wait*( ) have associated with the condition variable during their waits; however, if predictable scheduling behavior is required, then that mutex shall be locked by the thread calling *pthread_cond_broadcast*( ) or *pthread_cond_signal*( ).

The *pthread_cond_broadcast*( ) and *pthread_cond_signal*( ) functions shall have no effect if they determine that there are no threads blocked on *cond*.

The behavior is undefined if the value specified by the *cond* argument to *pthread_cond_broadcast*( ) or *pthread_cond_signal*( ) does not refer to an initialized condition variable.

RETURN VALUE

If successful, the *pthread_cond_broadcast*( ) and *pthread_cond_signal*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

The *pthread_cond_broadcast*( ) function is used whenever the shared-variable state has been changed in a way that more than one thread can proceed with its task. Consider a single producer/multiple consumer problem, where the producer can insert multiple items on a list that is accessed one item at a time by the consumers. By calling the *pthread_cond_broadcast*( ) function, the producer would notify all consumers that might be waiting, and thereby the application would receive more throughput on a multi-processor. In addition, *pthread_cond_broadcast*( ) makes it easier to implement a read-write lock. The *pthread_cond_broadcast*( ) function is needed in order to wake up all waiting readers when a

writer releases its lock. Finally, the two-phase commit algorithm can use this broadcast function to notify all clients of an impending transaction commit.

It is not safe to use the *pthread_cond_signal*( ) function in a signal handler that is invoked asynchronously. Even if it were safe, there would still be a race between the test of the Boolean *pthread_cond_wait*( ) that could not be efficiently eliminated.

Mutexes and condition variables are thus not suitable for releasing a waiting thread by signaling from code running in a signal handler.

RATIONALE

If an implementation detects that the value specified by the *cond* argument to *pthread_cond_broadcast*( ) or *pthread_cond_signal*( ) does not refer to an initialized condition variable, it is recommended that the function should fail and report an [EINVAL] error.

Multiple Awakenings by Condition Signal

On a multi-processor, it may be impossible for an implementation of *pthread_cond_signal*( ) to avoid the unblocking of more than one thread blocked on a condition variable. For example, consider the following partial implementation of *pthread_cond_wait*( ) and *pthread_cond_signal*( ), executed by two threads in the order given. One thread is trying to wait on the condition variable, another is concurrently executing *pthread_cond_signal*( ), while a third thread is already waiting.

```
pthread_cond_wait(mutex, cond):
    value = cond->value; /* 1 */
    pthread_mutex_unlock(mutex); /* 2 */
    pthread_mutex_lock(cond->mutex); /* 10 */
    if (value == cond->value) { /* 11 */
        me->next_cond = cond->waiter;
        cond->waiter = me;
        pthread_mutex_unlock(cond->mutex);
        unable_to_run(me);
    } else
        pthread_mutex_unlock(cond->mutex); /* 12 */
    pthread_mutex_lock(mutex); /* 13 */

pthread_cond_signal(cond):
    pthread_mutex_lock(cond->mutex); /* 3 */
    cond->value++; /* 4 */
    if (cond->waiter) { /* 5 */
        sleeper = cond->waiter; /* 6 */
        cond->waiter = sleeper->next_cond; /* 7 */
        able_to_run(sleeper); /* 8 */
    }
    pthread_mutex_unlock(cond->mutex); /* 9 */
```

The effect is that more than one thread can return from its call to *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), or *pthread_cond_wait*( ) as a result of one call to *pthread_cond_signal*( ). This effect is called ``spurious wakeup''. Note that the situation is self-correcting in that the number of threads that are so awakened is finite; for example, the next thread to call *pthread_cond_wait*( ) after the sequence of events above blocks.

While this problem could be resolved, the loss of efficiency for a fringe condition that occurs only rarely is unacceptable, especially given that one has to check the predicate associated with a condition variable anyway. Correcting this problem would unnecessarily reduce the degree of

concurrency in this basic building block for all higher-level synchronization operations.

An added benefit of allowing spurious wakeups is that applications are forced to code a predicate-testing-loop around the condition wait. This also makes the application tolerate superfluous condition broadcasts or signals on the same condition variable that may be coded in some other part of the application. The resulting applications are thus more robust. Therefore, POSIX.1-2024 explicitly documents that spurious wakeups may occur.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_clockwait( ), *pthread_cond_destroy*( )

XBD Section 4.15.2 (on page 104), **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_cond_broadcast*( ) and *pthread_cond_signal*( ) functions are marked as part of the Threads option.

The APPLICATION USAGE section is added.

Issue 7

The *pthread_cond_broadcast*( ) and *pthread_cond_signal*( ) functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized condition variable is removed; this condition results in undefined behavior.

Issue 8

Austin Group Defect 609 is applied, adding atomicity requirements.

Austin Group Defect 1216 is applied, adding *pthread_cond_clockwait*( ).

NAME

pthread_cond_clockwait, pthread_cond_timedwait, pthread_cond_wait — wait on a condition

SYNOPSIS

```
#include <pthread.h>

int pthread_cond_clockwait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex, clockid_t clock_id,
    const struct timespec *restrict abstime);
int pthread_cond_timedwait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex,
    const struct timespec *restrict abstime);
int pthread_cond_wait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex);
```

DESCRIPTION

The *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), and *pthread_cond_wait*( ) functions shall block on a condition variable. The application shall ensure that these functions are called with *mutex* locked by the calling thread; otherwise, an error (for PTHREAD_MUTEX_ERRORCHECK and robust mutexes) or undefined behavior (for other mutexes) results.

These functions atomically release *mutex* and cause the calling thread to block on the condition variable *cond*; atomically here means ``atomically with respect to access by another thread to the mutex and then the condition variable''. That is, if another thread is able to acquire the mutex after the about-to-block thread has released it, then a subsequent call to *pthread_cond_broadcast*( ) or *pthread_cond_signal*( ) in that thread shall behave as if it were issued after the about-to-block thread has blocked.

Upon successful return, the mutex shall have been locked and shall be owned by the calling thread.

If *mutex* is a robust mutex where an owner terminated while holding the lock and the state is recoverable, the mutex shall be acquired even though the function returns [EOWNERDEAD].

When using condition variables there is always a Boolean predicate involving shared variables associated with each condition wait that is true if the thread should proceed. Spurious wakeups from the *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), or *pthread_cond_wait*( ) functions may occur. Since the return from *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), or *pthread_cond_wait*( ) does not imply anything about the value of this predicate, the predicate should be re-evaluated upon such return.

When a thread waits on a condition variable, having specified a particular mutex to the *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), or *pthread_cond_wait*( ) operation, a dynamic binding is formed between that mutex and condition variable that remains in effect as long as at least one thread is blocked on the condition variable. During this time, the effect of an attempt by any thread to wait on that condition variable using a different mutex is undefined. Once all waiting threads have been unblocked (as by the *pthread_cond_broadcast*( ) operation), the next wait operation on that condition variable shall form a new dynamic binding with the mutex specified by that wait operation. Even though the dynamic binding between condition variable and mutex may be removed or replaced between the time a thread is unblocked from a wait on the condition variable and the time that it returns to the caller or begins cancellation cleanup, the unblocked thread shall always re-acquire the mutex specified in the condition wait operation call from which it is returning.

A condition wait (whether timed or not) is a cancellation point. When the cancelability type of a thread is set to PTHREAD_CANCEL_DEFERRED, a side-effect of acting upon a cancellation request while in a condition wait is that the mutex is (in effect) re-acquired before calling the first

cancellation cleanup handler. The effect is as if the thread were unblocked, allowed to execute up to the point of returning from the call to *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), or *pthread_cond_wait*( ), but at that point notices the cancellation request and, instead of returning to the caller, starts the thread cancellation activities, which includes calling cancellation cleanup handlers.

A thread that has been unblocked because it has been canceled while blocked in a call to *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), or *pthread_cond_wait*( ) shall not consume any condition signal that may be directed concurrently at the condition variable if there are other threads blocked on the condition variable.

The *pthread_cond_clockwait*( ) function shall be equivalent to *pthread_cond_wait*( ), except that an error is returned if the absolute time specified by *abstime* as measured against the clock indicated by *clock_id* passes (that is, the current time measured by that clock equals or exceeds *abstime*) before the condition *cond* is signaled or broadcasted, or if the absolute time specified by *abstime* has already been passed at the time of the call. Implementations shall support passing CLOCK_REALTIME and CLOCK_MONOTONIC to *pthread_cond_clockwait*( ) as the *clock_id* argument. When such timeouts occur, *pthread_cond_clockwait*( ) shall nonetheless release and re-acquire the mutex referenced by *mutex*, and may consume a condition signal directed concurrently at the condition variable.

The *pthread_cond_timedwait*( ) function shall be equivalent to *pthread_cond_clockwait*( ), except that it lacks the *clock_id* argument. The clock to measure *abstime* against shall instead come from the condition variable's clock attribute which can be set by *pthread_condattr_setclock*( ) prior to the condition variable's creation. If no clock attribute has been set, the default shall be CLOCK_REALTIME.

If a signal is delivered to a thread waiting for a condition variable, upon return from the signal handler the thread resumes waiting for the condition variable as if it was not interrupted, or it shall return zero due to spurious wakeup.

The behavior is undefined if the value specified by the *cond* or *mutex* argument to these functions does not refer to an initialized condition variable or an initialized mutex object, respectively.

RETURN VALUE

Except for [ETIMEDOUT], [ENOTRECOVERABLE], and [EOWNERDEAD], all these error checks shall act as if they were performed immediately at the beginning of processing for the function and shall cause an error return, in effect, prior to modifying the state of the mutex specified by *mutex* or the condition variable specified by *cond*.

Upon successful completion, a value of zero shall be returned; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions shall fail if:

[EAGAIN] The mutex is a robust mutex and the system resources available for robust mutexes owned would be exceeded.

[ENOTRECOVERABLE]
The state protected by the mutex is not recoverable.

[EOWNERDEAD]
The mutex is a robust mutex and the process containing the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state

consistent.

[EPERM] The mutex type is PTHREAD_MUTEX_ERRORCHECK or the mutex is a robust mutex, and the current thread does not own the mutex.

The *pthread_cond_clockwait*( ) and *pthread_cond_timedwait*( ) functions shall fail if:

[ETIMEDOUT] The time specified by *abstime* has passed.

[EINVAL] The *abstime* argument specified a nanosecond value less than zero or greater than or equal to 1000 million, or the *clock_id* argument passed to *pthread_cond_clockwait*( ) is invalid or not supported.

These functions may fail if:

[EOWNERDEAD]
The mutex is a robust mutex and the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications that have assumed that non-zero return values are errors will need updating for use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If an application is supposed to work with normal and robust mutexes, it should check all return values for error conditions and if necessary take appropriate action.

RATIONALE

If an implementation detects that the value specified by the *cond* argument to *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), or *pthread_cond_wait*( ) does not refer to an initialized condition variable, or detects that the value specified by the *mutex* argument does not refer to an initialized mutex object, it is recommended that the function should fail and report an [EINVAL] error.

Condition Wait Semantics

It is important to note that when *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), and *pthread_cond_wait*( ) return without error, the associated predicate may still be false. Similarly, when *pthread_cond_clockwait*( ) or *pthread_cond_timedwait*( ) returns with the timeout error, the associated predicate may be true due to an unavoidable race between the expiration of the timeout and the predicate state change.

The application needs to recheck the predicate on any return because it cannot be sure there is another thread waiting on the thread to handle the signal, and if there is not then the signal is lost. The burden is on the application to check the predicate.

Some implementations, particularly on a multi-processor, may sometimes cause multiple threads to wake up when the condition variable is signaled simultaneously on different processors.

In general, whenever a condition wait returns, the thread has to re-evaluate the predicate associated with the condition wait to determine whether it can safely proceed, should wait again, or should declare a timeout. A return from the wait does not imply that the associated

predicate is either true or false.

It is thus recommended that a condition wait be enclosed in the equivalent of a ``while loop'' that checks the predicate.

Timed Wait Semantics

An absolute time measure was chosen for specifying the timeout parameter for two reasons. First, a relative time measure can be easily implemented on top of a function that specifies absolute time, but there is a race condition associated with specifying an absolute timeout on top of a function that specifies relative timeouts. For example, assume that *clock_gettime*( ) returns the current time and *cond_relative_timed_wait*( ) uses relative timeouts:

```
clock_gettime(CLOCK_REALTIME, &now)
reltime = sleep_til_this_absolute_time –now;
cond_relative_timed_wait(c, m, &reltime);
```

If the thread is preempted between the first statement and the last statement, the thread blocks for too long. Blocking, however, is irrelevant if an absolute timeout is used. An absolute timeout also need not be recomputed if it is used multiple times in a loop, such as that enclosing a condition wait.

For cases when the system clock is advanced discontinuously by an operator, it is expected that implementations process any timed wait expiring at an intervening time as if that time had actually occurred.

Choice of Clock

Care should be taken to decide which clock is most appropriate when waiting with a timeout. The system clock CLOCK_REALTIME, as used by default with *pthread_cond_timedwait*( ), may be subject to jumps forwards and backwards in order to correct it against actual time. CLOCK_MONOTONIC is guaranteed not to jump backwards and must also advance in real time, so using it via *pthread_cond_clockwait*( ) or *pthread_condattr_setclock*( ) may be more appropriate.

Cancellation and Condition Wait

A condition wait, whether timed or not, is a cancellation point. That is, the functions *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), and *pthread_cond_wait*( ) are points where a pending (or concurrent) cancellation request is noticed. The reason for this is that an indefinite wait is possible at these points—whatever event is being waited for, even if the program is totally correct, might never occur; for example, some input data being awaited might never be sent. By making condition wait a cancellation point, the thread can be canceled and perform its cancellation cleanup handler even though it may be stuck in some indefinite wait.

A side-effect of acting on a cancellation request while a thread is blocked on a condition variable is to re-acquire the mutex before calling any of the cancellation cleanup handlers. This is done in order to ensure that the cancellation cleanup handler is executed in the same state as the critical code that lies both before and after the call to the condition wait function. This rule is also required when interfacing to POSIX threads from languages, such as Ada or C++, which may choose to map cancellation onto a language exception; this rule ensures that each exception handler guarding a critical section can always safely depend upon the fact that the associated mutex has already been locked regardless of exactly where within the critical section the exception was raised. Without this rule, there would not be a uniform rule that exception handlers could follow regarding the lock, and so coding would become very cumbersome.

Therefore, since *some* statement has to be made regarding the state of the lock when a cancellation is delivered during a wait, a definition has been chosen that makes application coding most convenient and error free.

When acting on a cancellation request while a thread is blocked on a condition variable, the implementation is required to ensure that the thread does not consume any condition signals directed at that condition variable if there are any other threads waiting on that condition variable. This rule is specified in order to avoid deadlock conditions that could occur if these two independent requests (one acting on a thread and the other acting on the condition variable) were not processed independently.

Performance of Mutexes and Condition Variables

Mutexes are expected to be locked only for a few instructions. This practice is almost automatically enforced by the desire of programmers to avoid long serial regions of execution (which would reduce total effective parallelism).

When using mutexes and condition variables, one tries to ensure that the usual case is to lock the mutex, access shared data, and unlock the mutex. Waiting on a condition variable should be a relatively rare situation. For example, when implementing a read-write lock, code that acquires a read-lock typically needs only to increment the count of readers (under mutual-exclusion) and return. The calling thread would actually wait on the condition variable only when there is already an active writer. So the efficiency of a synchronization operation is bounded by the cost of mutex lock/unlock and not by condition wait. Note that in the usual case there is no context switch.

This is not to say that the efficiency of condition waiting is unimportant. Since there needs to be at least one context switch per Ada rendezvous, the efficiency of waiting on a condition variable is important. The cost of waiting on a condition variable should be little more than the minimal cost for a context switch plus the time to unlock and lock the mutex.

Features of Mutexes and Condition Variables

It had been suggested that the mutex acquisition and release be decoupled from condition wait. This was rejected because it is the combined nature of the operation that, in fact, facilitates realtime implementations. Those implementations can atomically move a high-priority thread between the condition variable and the mutex in a manner that is transparent to the caller. This can prevent extra context switches and provide more deterministic acquisition of a mutex when the waiting thread is signaled. Thus, fairness and priority issues can be dealt with directly by the scheduling discipline. Furthermore, the current condition wait operation matches existing practice.

Scheduling Behavior of Mutexes and Condition Variables

Synchronization primitives that attempt to interfere with scheduling policy by specifying an ordering rule are considered undesirable. Threads waiting on mutexes and condition variables are selected to proceed in an order dependent upon the scheduling policy rather than in some fixed order (for example, FIFO or priority). Thus, the scheduling policy determines which thread(s) are awakened and allowed to proceed.

Timed Condition Wait

The *pthread_cond_clockwait*( ) and *pthread_cond_timedwait*( ) functions allow an application to give up waiting for a particular condition after a given amount of time. An example follows:

```
(void) pthread_mutex_lock(&t.mn);
    t.waiters++;
    clock_gettime(CLOCK_MONOTONIC, &ts);
    ts.tv_sec += 5;
    rc = 0;
    while (! mypredicate(&t) && rc == 0)
        rc = pthread_cond_clockwait(&t.cond, &t.mn,
            CLOCK_MONOTONIC, &ts);
    t.waiters--;
    if (rc == 0 || mypredicate(&t))
        setmystate(&t);
(void) pthread_mutex_unlock(&t.mn);
```

By making the timeout parameter absolute, it does not need to be recomputed each time the program checks its blocking predicate. If the timeout was relative, it would have to be recomputed before each call. This would be especially difficult since such code would need to take into account the possibility of extra wakeups that result from extra broadcasts or signals on the condition variable that occur before either the predicate is true or the timeout is due. Using CLOCK_MONOTONIC rather than CLOCK_REALTIME means that the timeout is not influenced by the system clock being changed.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_broadcast( )

XBD Section 4.15.2 (on page 104), **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_cond_timedwait*( ) and *pthread_cond_wait*( ) functions are marked as part of the Threads option.

The Open Group Corrigendum U021/9 is applied, correcting the prototype for the *pthread_cond_wait*( ) function.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for the Clock Selection option.

The ERRORS section has an additional case for [EPERM] in response to IEEE PASC Interpretation 1003.1c #28.

The **restrict** keyword is added to the *pthread_cond_timedwait*( ) and *pthread_cond_wait*( ) prototypes for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/89 is applied, updating the DESCRIPTION for consistency with the *pthread_cond_destroy*( ) function that states it is safe to destroy an initialized condition variable upon which no threads are currently blocked.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/90 is applied, updating words in the DESCRIPTION from ``the cancelability enable state'' to ``the cancelability type''.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/91 is applied, updating the ERRORS section to remove the error case related to *abstime* from the *pthread_cond_wait*( ) function, and to make the error case related to *abstime* mandatory for *pthread_cond_timedwait*( ) for consistency with other functions.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/92 is applied, adding a new paragraph to the RATIONALE section stating that an application should check the predicate on any return from this function.

Issue 7

SD5-XSH-ERN-44 is applied, changing the definition of the ``shall fail'' case of the [EINVAL] error.

Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

The *pthread_cond_timedwait*( ) and *pthread_cond_wait*( ) functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized condition variable or uninitialized mutex object is removed; this condition results in undefined behavior"

The [EPERM] error is revised and moved to the ``shall fail'' list of error conditions for the *pthread_cond_timedwait*( ) function.

The DESCRIPTION is updated to clarify the behavior when *mutex* is a robust mutex.

The ERRORS section is updated to include ``shall fail'' cases for PTHREAD_MUTEX_ERRORCHECK mutexes.

The DESCRIPTION is rewritten to clarify that undefined behavior occurs only for mutexes where the [EPERM] error is not mandated.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0456 [91,286,437] and XSH/TC1-2008/0457 [239] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0271 [749] is applied.

Issue 8

Austin Group Defect 354 is applied, adding the [EAGAIN] error.

Austin Group Defect 1162 is applied, changing ``an error code'' to ``[EOWNERDEAD]''.

Austin Group Defects 1216 and 1485 are applied, adding *pthread_cond_clockwait*( ).

NAME

pthread_cond_destroy, pthread_cond_init — destroy and initialize condition variables

SYNOPSIS

```
#include <pthread.h>

int pthread_cond_destroy(pthread_cond_t *cond);
int pthread_cond_init(pthread_cond_t *restrict cond,
    const pthread_condattr_t *restrict attr);
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

DESCRIPTION

The *pthread_cond_destroy*( ) function shall destroy the given condition variable specified by *cond*; the object becomes, in effect, uninitialized. An implementation may cause *pthread_cond_destroy*( ) to set the object referenced by *cond* to an invalid value. A destroyed condition variable object can be reinitialized using *pthread_cond_init*( ); the results of otherwise referencing the object after it has been destroyed are undefined.

It shall be safe to destroy an initialized condition variable upon which no threads are currently blocked. Attempting to destroy a condition variable upon which other threads are currently blocked results in undefined behavior.

The *pthread_cond_init*( ) function shall initialize the condition variable referenced by *cond* with attributes referenced by *attr*. If *attr* is NULL, the default condition variable attributes shall be used; the effect is the same as passing the address of a default condition variable attributes object. Upon successful initialization, the state of the condition variable shall become initialized.

See Section 2.9.9 (on page 548) for further requirements.

Attempting to initialize an already initialized condition variable results in undefined behavior.

In cases where default condition variable attributes are appropriate, the macro PTHREAD_COND_INITIALIZER can be used to initialize condition variables. The effect shall be equivalent to dynamic initialization by a call to *pthread_cond_init*( ) with parameter *attr* specified as NULL, except that no error checks are performed.

The behavior is undefined if the value specified by the *cond* argument to *pthread_cond_destroy*( ) does not refer to an initialized condition variable.

The behavior is undefined if the value specified by the *attr* argument to *pthread_cond_init*( ) does not refer to an initialized condition variable attributes object.

RETURN VALUE

If successful, the *pthread_cond_destroy*( ) and *pthread_cond_init*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_cond_init*( ) function shall fail if:

[EAGAIN] The system lacked the necessary resources (other than memory) to initialize another condition variable.

[ENOMEM] Insufficient memory exists to initialize the condition variable.

These functions shall not return an error code of [EINTR].

EXAMPLES

A condition variable can be destroyed immediately after all the threads that are blocked on it are awakened. For example, consider the following code:

```
struct list {
    pthread_mutex_t lm;
    ...
}

struct elt {
    key k;
    int busy;
    pthread_cond_t notbusy;
    ...
}

/* Find a list element and reserve it. */
struct elt *
list_find(struct list *lp, key k)
{
    struct elt *ep;

    pthread_mutex_lock(&lp->lm);
    while ((ep = find_elt(l, k) != NULL) && ep->busy)
        pthread_cond_wait(&ep->notbusy, &lp->lm);
    if (ep != NULL)
        ep->busy = 1;
    pthread_mutex_unlock(&lp->lm);
    return(ep);
}
delete_elt(struct list *lp, struct elt *ep)
{
    pthread_mutex_lock(&lp->lm);
    assert(ep->busy);
    ... remove ep from list ...
    ep->busy = 0;  /* Paranoid. */
(A) pthread_cond_broadcast(&ep->notbusy);
    pthread_mutex_unlock(&lp->lm);
(B) pthread_cond_destroy(&ep->notbusy);
    free(ep);
}
```

In this example, the condition variable and its list element may be freed (line B) immediately after all threads waiting for it are awakened (line A), since the mutex and the code ensure that no other thread can touch the element to be deleted.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *cond* argument to *pthread_cond_destroy*( ) does not refer to an initialized condition variable, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the *cond* argument to *pthread_cond_destroy*( ) or *pthread_cond_init*( ) refers to a condition variable that is in use (for

example, in a *pthread_cond_wait*( ) call) by another thread, or detects that the value specified by the *cond* argument to *pthread_cond_init*( ) refers to an already initialized condition variable, it is recommended that the function should fail and report an [EBUSY] error.

If an implementation detects that the value specified by the *attr* argument to *pthread_cond_init*( ) does not refer to an initialized condition variable attributes object, it is recommended that the function should fail and report an [EINVAL] error.

See also *pthread_mutex_destroy*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_broadcast( ), *pthread_cond_clockwait*( ), *pthread_mutex_destroy*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_cond_destroy*( ) and *pthread_cond_init*( ) functions are marked as part of the Threads option.

IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

The **restrict** keyword is added to the *pthread_cond_init*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

The *pthread_cond_destroy*( ) and *pthread_cond_init*( ) functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized condition variable and an uninitialized condition variable attributes object is removed; this condition results in undefined behavior.

The [EBUSY] error for a condition variable already in use or an already initialized condition variable is removed; this condition results in undefined behavior.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0455 [70] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0269 [972] and XSH/TC2-2008/0270 [910] are applied.

NAME

pthread_cond_signal — signal a condition

SYNOPSIS

```
#include <pthread.h>

int pthread_cond_signal(pthread_cond_t *cond);
```

DESCRIPTION

Refer to *pthread_cond_broadcast( )*.

56791 **NAME**

56792 pthread_cond_timedwait, pthread_cond_wait — wait on a condition

56793 **SYNOPSIS**

56794
```
#include <pthread.h>
```

56795
```
int pthread_cond_timedwait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex,
    const struct timespec *restrict abstime);
int pthread_cond_wait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex);
```

56800 **DESCRIPTION**

56801 Refer to *pthread_cond_clockwait( )*.

NAME

pthread_condattr_destroy, pthread_condattr_init — destroy and initialize the condition variable attributes object

SYNOPSIS

```
#include <pthread.h>

int pthread_condattr_destroy(pthread_condattr_t *attr);
int pthread_condattr_init(pthread_condattr_t *attr);
```

DESCRIPTION

The *pthread_condattr_destroy*( ) function shall destroy a condition variable attributes object; the object becomes, in effect, uninitialized. An implementation may cause *pthread_condattr_destroy*( ) to set the object referenced by *attr* to an invalid value. A destroyed *attr* attributes object can be reinitialized using *pthread_condattr_init*( ); the results of otherwise referencing the object after it has been destroyed are undefined.

The *pthread_condattr_init*( ) function shall initialize a condition variable attributes object *attr* with the default value for all of the attributes defined by the implementation.

Results are undefined if *pthread_condattr_init*( ) is called specifying an already initialized *attr* attributes object.

After a condition variable attributes object has been used to initialize one or more condition variables, any function affecting the attributes object (including destruction) shall not affect any previously initialized condition variables.

This volume of POSIX.1-2024 requires two attributes, the *clock* attribute and the *process-shared* attribute.

Additional attributes, their default values, and the names of the associated functions to get and set those attribute values are implementation-defined.

The behavior is undefined if the value specified by the *attr* argument to *pthread_condattr_destroy*( ) does not refer to an initialized condition variable attributes object.

RETURN VALUE

If successful, the *pthread_condattr_destroy*( ) and *pthread_condattr_init*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_condattr_init*( ) function shall fail if:

[ENOMEM] Insufficient memory exists to initialize the condition variable attributes object.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

A *process-shared* attribute has been defined for condition variables for the same reason it has been defined for mutexes.

If an implementation detects that the value specified by the *attr* argument to *pthread_condattr_destroy*( ) does not refer to an initialized condition variable attributes object, it is recommended that the function should fail and report an [EINVAL] error.

See also *pthread_attr_destroy*( ) and *pthread_mutex_destroy*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_cond_destroy*( ), *pthread_condattr_getpshared*( ), *pthread_create*( ), *pthread_mutex_destroy*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_condattr_destroy*( ) and *pthread_condattr_init*( ) functions are marked as part of the Threads option.

Issue 7

The *pthread_condattr_destroy*( ) and *pthread_condattr_init*( ) functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized condition variable attributes object is removed; this condition results in undefined behavior.

NAME

pthread_condattr_getclock, pthread_condattr_setclock — get and set the clock selection condition variable attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_condattr_getclock(const pthread_condattr_t *restrict attr,
    clockid_t *restrict clock_id);
int pthread_condattr_setclock(pthread_condattr_t *attr,
    clockid_t clock_id);
```

DESCRIPTION

The *pthread_condattr_getclock*( ) function shall obtain the value of the *clock* attribute from the attributes object referenced by *attr*.

The *pthread_condattr_setclock*( ) function shall set the *clock* attribute in an initialized attributes object referenced by *attr*. If *pthread_condattr_setclock*( ) is called with a *clock_id* argument that refers to a CPU-time clock, the call shall fail.

The *clock* attribute is the clock ID of the clock that shall be used to measure the timeout service of *pthread_cond_timedwait*( ). The default value of the *clock* attribute shall refer to the system clock. The *clock* attribute shall have no effect on the *pthread_cond_clockwait*( ) function.

The behavior is undefined if the value specified by the *attr* argument to *pthread_condattr_getclock*( ) or *pthread_condattr_setclock*( ) does not refer to an initialized condition variable attributes object.

RETURN VALUE

If successful, the *pthread_condattr_getclock*( ) function shall return zero and store the value of the clock attribute of *attr* into the object referenced by the *clock_id* argument. Otherwise, an error number shall be returned to indicate the error.

If successful, the *pthread_condattr_setclock*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_condattr_setclock*( ) function may fail if:

[EINVAL] The value specified by *clock_id* does not refer to a known clock, or is a CPU-time clock.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_condattr_getclock*( ) or *pthread_condattr_setclock*( ) does not refer to an initialized condition variable attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_clockwait( ), *pthread_cond_destroy*( ), *pthread_condattr_destroy*( ), *pthread_condattr_getpshared*( ), *pthread_create*( ), *pthread_mutex_destroy*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7

The *pthread_condattr_getclock*( ) and *pthread_condattr_setclock*( ) functions are moved from the Clock Selection option to the Base.

The [EINVAL] error for an uninitialized condition variable attributes object is removed; this condition results in undefined behavior.

Issue 8

Austin Group Defect 1216 is applied, adding *pthread_cond_clockwait*( ).

NAME

pthread_condattr_getpshared, pthread_condattr_setpshared — get and set the process-shared condition variable attributes

SYNOPSIS

TSH
```
#include <pthread.h>

int pthread_condattr_getpshared(const pthread_condattr_t *restrict attr,
    int *restrict pshared);
int pthread_condattr_setpshared(pthread_condattr_t *attr,
    int pshared);
```

DESCRIPTION

The *pthread_condattr_getpshared*( ) function shall obtain the value of the *process-shared* attribute from the attributes object referenced by *attr*.

The *pthread_condattr_setpshared*( ) function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

The *process-shared* attribute is set to PTHREAD_PROCESS_SHARED to permit a condition variable to be operated upon by any thread that has access to the memory where the condition variable is allocated, even if the condition variable is allocated in memory that is shared by multiple processes. See Section 2.9.9 (on page 548) for further requirements. The default value of the attribute is PTHREAD_PROCESS_PRIVATE.

The behavior is undefined if the value specified by the *attr* argument to *pthread_condattr_getpshared*( ) or *pthread_condattr_setpshared*( ) does not refer to an initialized condition variable attributes object.

RETURN VALUE

If successful, the *pthread_condattr_setpshared*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

If successful, the *pthread_condattr_getpshared*( ) function shall return zero and store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_condattr_setpshared*( ) function may fail if:

[EINVAL] The new value specified for the attribute is outside the range of legal values for that attribute.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_condattr_getpshared*( ) or *pthread_condattr_setpshared*( ) does not refer to an initialized condition variable attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_create( ), *pthread_cond_destroy*( ), *pthread_condattr_destroy*( ), *pthread_mutex_destroy*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_condattr_getpshared*( ) and *pthread_condattr_setpshared*( ) functions are marked as part of the Threads and Thread Process-Shared Synchronization options.

The **restrict** keyword is added to the *pthread_condattr_getpshared*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

The *pthread_condattr_getpshared*( ) and *pthread_condattr_setpshared*( ) functions are marked only as part of the Thread Process-Shared Synchronization option as the Threads option is now part of the Base.

The [EINVAL] error for an uninitialized condition variable attributes object is removed; this condition results in undefined behavior.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0272 [972] and XSH/TC2-2008/0273 [757] are applied.

56980 **NAME**

56981 pthread_condattr_init — initialize the condition variable attributes object

56982 **SYNOPSIS**

56983 `#include <pthread.h>`

56984 `int pthread_condattr_init(pthread_condattr_t *attr);`

56985 **DESCRIPTION**

56986 Refer to *pthread_condattr_destroy( )*.

NAME

pthread_condattr_setclock — set the clock selection condition variable attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_condattr_setclock(pthread_condattr_t *attr,
    clockid_t clock_id);
```

DESCRIPTION

Refer to *pthread_condattr_getclock( )*.

NAME

pthread_condattr_setpshared — set the process-shared condition variable attribute

SYNOPSIS

TSH

```
#include <pthread.h>

int pthread_condattr_setpshared(pthread_condattr_t *attr,
    int pshared);
```

DESCRIPTION

Refer to *pthread_condattr_getpshared*( ).

57003 **NAME**

57004 pthread_create — thread creation

57005 **SYNOPSIS**

57006 ```
#include <pthread.h>
```

57007 ```
int pthread_create(pthread_t *restrict thread,
```
57008 ```
    const pthread_attr_t *restrict attr,
```
57009 ```
    void *(*start_routine)(void*), void *restrict arg);
```

57010 **DESCRIPTION**

57011 The *pthread_create*( ) function shall create a new thread, with attributes specified by *attr*, within a
57012 process. If *attr* is NULL, the default attributes shall be used. If the attributes specified by *attr* are
57013 modified later, the thread's attributes shall not be affected. Upon successful completion,
57014 *pthread_create*( ) shall store the ID of the created thread in the location referenced by *thread*.

57015 The thread is created executing *start_routine* with *arg* as its sole argument. If the *start_routine*
57016 returns, the effect shall be as if there was an implicit call to *pthread_exit*( ) using the return value
57017 of *start_routine* as the exit status. Note that the thread in which *main*( ) was originally invoked
57018 differs from this. When it returns from *main*( ), the effect shall be as if there was an implicit call to
57019 *exit*( ) using the return value of *main*( ) as the exit status.

57020 The signal state of the new thread shall be initialized as follows:

57021 • The signal mask shall be inherited from the creating thread.

57022 • The set of signals pending for the new thread shall be empty.

57023 XSI The thread-local current locale and the alternate stack shall not be inherited.

57024 The floating-point environment shall be inherited from the creating thread.

57025 If *pthread_create*( ) fails, no new thread is created and the contents of the location referenced by
57026 *thread* are undefined.

57027 TCT If _POSIX_THREAD_CPUTIME is defined, the new thread shall have a CPU-time clock
57028 accessible, and the initial value of this clock shall be set to zero.

57029 The behavior is undefined if the value specified by the *attr* argument to *pthread_create*( ) does not
57030 refer to an initialized thread attributes object.

57031 **RETURN VALUE**

57032 If successful, the *pthread_create*( ) function shall return zero; otherwise, an error number shall be
57033 returned to indicate the error.

57034 **ERRORS**

57035 The *pthread_create*( ) function shall fail if:

57036 [EAGAIN] The system lacked the necessary resources to create another thread, or the
57037 system-imposed limit on the total number of threads in a process
57038 {PTHREAD_THREADS_MAX} would be exceeded.

57039 [EPERM] The caller does not have appropriate privileges to set the required scheduling
57040 parameters or scheduling policy.

57041 The *pthread_create*( ) function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

There is no requirement on the implementation that the ID of the created thread be available before the newly created thread starts executing. The calling thread can obtain the ID of the created thread through the *thread* argument of the *pthread_create*( ) function, and the newly created thread can obtain its ID by a call to *pthread_self*( ).

RATIONALE

A suggested alternative to *pthread_create*( ) would be to define two separate operations: create and start. Some applications would find such behavior more natural. Ada, in particular, separates the ``creation'' of a task from its ``activation''.

Splitting the operation was rejected by the standard developers for many reasons:

- The number of calls required to start a thread would increase from one to two and thus place an additional burden on applications that do not require the additional synchronization. The second call, however, could be avoided by the additional complication of a start-up state attribute.
- An extra state would be introduced: ``created but not started''. This would require the standard to specify the behavior of the thread operations when the target has not yet started executing.
- For those applications that require such behavior, it is possible to simulate the two separate steps with the facilities that are currently provided. The *start_routine*( ) can synchronize by waiting on a condition variable that is signaled by the start operation.

An Ada implementor can choose to create the thread at either of two points in the Ada program: when the task object is created, or when the task is activated (generally at a ``begin''). If the first approach is adopted, the *start_routine*( ) needs to wait on a condition variable to receive the order to begin ``activation''. The second approach requires no such condition variable or extra synchronization. In either approach, a separate Ada task control block would need to be created when the task object is created to hold rendezvous queues, and so on.

An extension of the preceding model would be to allow the state of the thread to be modified between the create and start. This would allow the thread attributes object to be eliminated. This has been rejected because:

- All state in the thread attributes object has to be able to be set for the thread. This would require the definition of functions to modify thread attributes. There would be no reduction in the number of function calls required to set up the thread. In fact, for an application that creates all threads using identical attributes, the number of function calls required to set up the threads would be dramatically increased. Use of a thread attributes object permits the application to make one set of attribute setting function calls. Otherwise, the set of attribute setting function calls needs to be made for each thread creation.
- Depending on the implementation architecture, functions to set thread state would require kernel calls, or for other implementation reasons would not be able to be implemented as macros, thereby increasing the cost of thread creation.
- The ability for applications to segregate threads by class would be lost.

Another suggested alternative uses a model similar to that for process creation, such as ``thread fork''. The fork semantics would provide more flexibility and the ``create'' function can be implemented simply by doing a thread fork followed immediately by a call to the desired ``start

routine'' for the thread. This alternative has these problems:

- For many implementations, the entire stack of the calling thread would need to be duplicated, since in many architectures there is no way to determine the size of the calling frame.
- Efficiency is reduced since at least some part of the stack has to be copied, even though in most cases the thread never needs the copied context, since it merely calls the desired start routine.

If an implementation detects that the value specified by the *attr* argument to *pthread_create*( ) does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

fork( ), *pthread_exit*( ), *pthread_join*( )

XBD Section 4.15.2 (on page 104), **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_create*( ) function is marked as part of the Threads option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EPERM] mandatory error condition is added.

The thread CPU-time clock semantics are added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *pthread_create*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION is updated to make it explicit that the floating-point environment is inherited from the creating thread.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/44 is applied, adding text that the alternate stack is not inherited.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/93 is applied, updating the ERRORS section to remove the mandatory [EINVAL] error (``The value specified by *attr* is invalid''), and adding the optional [EINVAL] error (``The attributes specified by *attr* are invalid'').

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/94 is applied, adding the APPLICATION USAGE section.

Issue 7

The *pthread_create*( ) function is moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0458 [302] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0274 [849] is applied.

57128 **NAME**

57129 pthread_detach — detach a thread

57130 **SYNOPSIS**

57131 ```
#include <pthread.h>
```

57132 ```
int pthread_detach(pthread_t thread);
```

57133 **DESCRIPTION**

57134 The *pthread_detach*( ) function shall change the thread *thread* from joinable to detached, indicating
57135 to the implementation that storage for the thread can be reclaimed when the thread terminates.
57136 If *thread* has not terminated, *pthread_detach*( ) shall not cause it to terminate, but shall prevent the
57137 thread from becoming a zombie thread when it does terminate.

57138 The behavior is undefined if the value specified by the *thread* argument to *pthread_detach*( ) does
57139 not refer to a joinable thread.

57140 **RETURN VALUE**

57141 If the call succeeds, *pthread_detach*( ) shall return 0; otherwise, an error number shall be returned
57142 to indicate the error.

57143 **ERRORS**

57144 The *pthread_detach*( ) function shall not return an error code of [EINTR].

57145 **EXAMPLES**

57146 None.

57147 **APPLICATION USAGE**

57148 None.

57149 **RATIONALE**

57150 The *pthread_join*( ) or *pthread_detach*( ) functions should eventually be called for every thread that
57151 is created so that storage associated with the thread may be reclaimed.

57152 It has been suggested that a ``detach'' function is not necessary; the *detachstate* thread creation
57153 attribute is sufficient, since a thread need never be dynamically detached. However, need arises
57154 in at least two cases:

57155 1. In a cancellation handler for a *pthread_join*( ) it is nearly essential to have a
57156 *pthread_detach*( ) function in order to detach the thread on which *pthread_join*( ) was
57157 waiting. Without it, it would be necessary to have the handler do another *pthread_join*( ) to
57158 attempt to detach the thread, which would both delay the cancellation processing for an
57159 unbounded period and introduce a new call to *pthread_join*( ), which might itself need a
57160 cancellation handler. A dynamic detach is nearly essential in this case.

57161 2. In order to detach the ``initial thread'' (as may be desirable in processes that set up server
57162 threads).

57163 If an implementation detects that the value specified by the *thread* argument to *pthread_detach*( )
57164 does not refer to a joinable thread, it is recommended that the function should fail and report an
57165 [EINVAL] error.

57166 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended
57167 that the function should fail and report an [ESRCH] error.

57168 **FUTURE DIRECTIONS**

57169 None.

SEE ALSO

pthread_join( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_detach*( ) function is marked as part of the Threads option.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/95 is applied, updating the ERRORS section so that the [EINVAL] and [ESRCH] error cases become optional.

Issue 7

The *pthread_detach*( ) function is moved from the Threads option to the Base.

Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

The [EINVAL] error for a non-joinable thread is removed; this condition results in undefined behavior.

Issue 8

Austin Group Defect 792 is applied, clarifying that detaching a live thread prevents it becoming a zombie thread when it terminates.

Austin Group Defect 1167 is applied, clarifying that a thread is no longer joinable after *pthread_detach*( ) has been called for it.

NAME

pthread_equal — compare thread IDs

SYNOPSIS

```
#include <pthread.h>

int pthread_equal(pthread_t t1, pthread_t t2);
```

DESCRIPTION

This function shall compare the thread IDs *t1* and *t2*.

RETURN VALUE

The *pthread_equal*( ) function shall return a non-zero value if *t1* and *t2* are equal; otherwise, zero shall be returned.

If either *t1* or *t2* is not a valid thread ID and is not equal to PTHREAD_NULL, the behavior is undefined.

ERRORS

No errors are defined.

The *pthread_equal*( ) function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Implementations may choose to define a thread ID as a structure. This allows additional flexibility and robustness over using an **int**. For example, a thread ID could include a sequence number that allows detection of ``dangling IDs'' (copies of a thread ID that has been detached). Since the C language does not support comparison on structure types, the *pthread_equal*( ) function is provided to compare thread IDs.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_create( ), *pthread_self*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_equal*( ) function is marked as part of the Threads option.

Issue 7

The *pthread_equal*( ) function is moved from the Threads option to the Base.

Issue 8

Austin Group Defect 599 is applied, changing the RETURN VALUE section to mention PTHREAD_NULL.

NAME

pthread_exit — thread termination

SYNOPSIS

```
#include <pthread.h>

_Noreturn void pthread_exit(void *value_ptr);
```

DESCRIPTION

The *pthread_exit*( ) function shall terminate the calling thread and make the value *value_ptr* available to any successful join with the terminating thread. Any cancellation cleanup handlers that have been pushed and not yet popped shall be popped in the reverse order that they were pushed and then executed. After all cancellation cleanup handlers have been executed, if the thread has any thread-specific data (whether associated with key type **tss_t** or **pthread_key_t**), appropriate destructor functions shall be called in an unspecified order. Thread termination does not release any application visible process resources, including, but not limited to, mutexes and file descriptors, nor does it perform any process-level cleanup actions, including, but not limited to, calling any *atexit*( ) routines that may exist.

An implicit call to *pthread_exit*( ) is made when a thread that was not created using *thrd_create*( ), and is not the thread in which *main*( ) was first invoked, returns from the start routine that was used to create it. The function's return value shall serve as the thread's exit status.

The behavior of *pthread_exit*( ) is undefined if called from a cancellation cleanup handler or destructor function that was invoked as a result of either an implicit or explicit call to *pthread_exit*( ).

After a thread has terminated, the result of access to local (auto) variables of the thread is undefined. Thus, references to local variables of the exiting thread should not be used for the *pthread_exit*( ) *value_ptr* parameter value.

The process shall exit with an exit status of 0 after the last thread has been terminated. The behavior shall be as if the implementation called *exit*( ) with a zero argument at thread termination time.

RETURN VALUE

The *pthread_exit*( ) function cannot return to its caller.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Calls to *pthread_exit*( ) should not be made from threads created using *thrd_create*( ), as their exit status has a different type (**int** instead of **void** *). If *pthread_exit*( ) is called from the initial thread and it is not the last thread to terminate, other threads should not try to obtain its exit status using *thrd_join*( ).

RATIONALE

The normal mechanism by which a thread that was started using *pthread_create*( ) terminates is to return from the routine that was specified in the *pthread_create*( ) call that started it. The *pthread_exit*( ) function provides the capability for a thread to terminate without requiring a return from the start routine of that thread, thereby providing a function analogous to *exit*( ).

Regardless of the method of thread termination, any cancellation cleanup handlers that have been pushed and not yet popped are executed, and the destructors for any existing thread-specific data are executed. This volume of POSIX.1-2024 requires that cancellation cleanup

handlers be popped and called in order. After all cancellation cleanup handlers have been executed, thread-specific data destructors are called, in an unspecified order, for each item of thread-specific data that exists in the thread. This ordering is necessary because cancellation cleanup handlers may rely on thread-specific data.

As the meaning of the status is determined by the application (except when the thread has been canceled, in which case it is PTHREAD_CANCELED), the implementation has no idea what an illegal status value is, which is why no address error checking is done.

FUTURE DIRECTIONS

None.

SEE ALSO

exit( ), *pthread_create*( ), *pthread_join*( ), *pthread_key_create*( ), *thrd_create*( ), *thrd_exit*( ), *tss_create*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_exit*( ) function is marked as part of the Threads option.

Issue 7

The *pthread_exit*( ) function is moved from the Threads option to the Base.

Issue 8

Austin Group Defect 1302 is applied, adding `_Noreturn` to the SYNOPSIS, and updating the page to account for the addition of **<threads.h>** interfaces.

NAME

pthread_getcpuclockid — access a thread CPU-time clock (**ADVANCED REALTIME THREADS**)

SYNOPSIS

TCT

```
#include <pthread.h>
#include <time.h>

int pthread_getcpuclockid(pthread_t thread_id, clockid_t *clock_id);
```

DESCRIPTION

The *pthread_getcpuclockid*( ) function shall return in *clock_id* the clock ID of the CPU-time clock of the thread specified by *thread_id*, if the thread specified by *thread_id* exists.

RETURN VALUE

Upon successful completion, *pthread_getcpuclockid*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *pthread_getcpuclockid*( ) function is part of the Thread CPU-Time Clocks option and need not be provided on all implementations.

RATIONALE

If an implementation detects use of a thread ID after the end of its lifetime, it is recommended that the function should fail and report an [ESRCH] error.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getcpuclockid( ), *clock_getres*( ), *timer_create*( )

XBD **<pthread.h>**, **<time.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

Issue 7

The *pthread_getcpuclockid*( ) function is marked only as part of the Thread CPU-Time Clocks option as the Threads option is now part of the Base.

Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0275 [757] is applied.

NAME

pthread_getschedparam, pthread_setschedparam — dynamic thread scheduling parameters access (**REALTIME THREADS**)

SYNOPSIS

TPS

```
#include <pthread.h>

int pthread_getschedparam(pthread_t thread, int *restrict policy,
    struct sched_param *restrict param);
int pthread_setschedparam(pthread_t thread, int policy,
    const struct sched_param *param);
```

DESCRIPTION

The *pthread_getschedparam*( ) and *pthread_setschedparam*( ) functions shall, respectively, get and set the scheduling policy and parameters of individual threads within a multi-threaded process to be retrieved and set. For SCHED_FIFO and SCHED_RR, the only required member of the **sched_param** structure is the priority *sched_priority*. For SCHED_OTHER, the affected scheduling parameters are implementation-defined.

The *pthread_getschedparam*( ) function shall retrieve the scheduling policy and scheduling parameters for the thread whose thread ID is given by *thread* and shall store those values in *policy* and *param*, respectively. The priority value returned from *pthread_getschedparam*( ) shall be the value specified by the most recent *pthread_setschedparam*( ), *pthread_setschedprio*( ), or *pthread_create*( ) call affecting the target thread. It shall not reflect any temporary adjustments to its priority as a result of any priority inheritance or ceiling functions. The *pthread_setschedparam*( ) function shall set the scheduling policy and associated scheduling parameters for the thread whose thread ID is given by *thread* to the policy and associated parameters provided in *policy* and *param*, respectively.

The *policy* parameter may have the value SCHED_OTHER, SCHED_FIFO, or SCHED_RR. The scheduling parameters for the SCHED_OTHER policy are implementation-defined. The SCHED_FIFO and SCHED_RR policies shall have a single scheduling parameter, *priority*.

TSP

If _POSIX_THREAD_SPORADIC_SERVER is defined, then the *policy* argument may have the value SCHED_SPORADIC, with the exception for the *pthread_setschedparam*( ) function that if the scheduling policy was not SCHED_SPORADIC at the time of the call, it is implementation-defined whether the function is supported; in other words, the implementation need not allow the application to dynamically change the scheduling policy to SCHED_SPORADIC. The sporadic server scheduling policy has the associated parameters *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, *sched_priority*, and *sched_ss_max_repl*. The specified *sched_ss_repl_period* shall be greater than or equal to the specified *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail. The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the function to succeed; if not, the function shall fail. It is unspecified whether the *sched_ss_repl_period* and *sched_ss_init_budget* values are stored as provided by this function or are rounded to align with the resolution of the clock being used.

If the *pthread_setschedparam*( ) function fails, the scheduling parameters shall not be changed for the target thread.

RETURN VALUE

If successful, the *pthread_getschedparam*( ) and *pthread_setschedparam*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_setschedparam*( ) function shall fail if:

[ENOTSUP] An attempt was made to set the policy or scheduling parameters to an unsupported value.

TSP [ENOTSUP] An attempt was made to dynamically change the scheduling policy to SCHED_SPORADIC, and the implementation does not support this change.

The *pthread_setschedparam*( ) function may fail if:

[EINVAL] The value specified by *policy* or one of the scheduling parameters associated with the scheduling policy *policy* is invalid.

[EPERM] The caller does not have appropriate privileges to set either the scheduling parameters or the scheduling policy of the specified thread.

[EPERM] The implementation does not allow the application to modify one of the parameters to the value specified.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects use of a thread ID after the end of its lifetime, it is recommended that the function should fail and report an [ESRCH] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_setschedprio( ), *sched_getparam*( ), *sched_getscheduler*( )

XBD **<pthread.h>**, **<sched.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_getschedparam*( ) and *pthread_setschedparam*( ) functions are marked as part of the Threads and Thread Execution Scheduling options.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Execution Scheduling option.

The Open Group Corrigendum U026/2 is applied, correcting the prototype for the *pthread_setschedparam*( ) function so that its second argument is of type **int**.

The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *pthread_getschedparam*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

The Open Group Corrigendum U047/1 is applied.

IEEE PASC Interpretation 1003.1 #96 is applied, noting that priority values can also be set by a call to the *pthread_setschedprio*( ) function.

Issue 7

The *pthread_getschedparam*( ) and *pthread_setschedparam*( ) functions are marked only as part of the Thread Execution Scheduling option as the Threads option is now part of the Base.

Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements for the *sched_ss_repl_period* and *sched_ss_init_budget* values.

Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0459 [314] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0276 [757] is applied.

57425 **NAME**

57426 pthread_getspecific, pthread_setspecific — thread-specific data management

57427 **SYNOPSIS**

57428 ```
#include <pthread.h>
```

57429 ```
void *pthread_getspecific(pthread_key_t key);
```

57430 ```
int pthread_setspecific(pthread_key_t key, const void *value);
```

57431 **DESCRIPTION**

57432 The *pthread_getspecific*( ) function shall return the value currently bound to the specified *key* on
57433 behalf of the calling thread.

57434 The *pthread_setspecific*( ) function shall associate a thread-specific *value* with a *key* obtained via a
57435 previous call to *pthread_key_create*( ). Different threads may bind different values to the same
57436 key. These values are typically pointers to blocks of dynamically allocated memory that have
57437 been reserved for use by the calling thread.

57438 The effect of calling *pthread_getspecific*( ) or *pthread_setspecific*( ) with a *key* value not obtained
57439 from *pthread_key_create*( ) or after *key* has been deleted with *pthread_key_delete*( ) is undefined.

57440 Both *pthread_getspecific*( ) and *pthread_setspecific*( ) may be called from a thread-specific data
57441 destructor function. A call to *pthread_getspecific*( ) for the thread-specific data key being
57442 destroyed shall return the value NULL, unless the value is changed (after the destructor starts)
57443 by a call to *pthread_setspecific*( ). Calling *pthread_setspecific*( ) from a thread-specific data
57444 destructor routine may result either in lost storage (after at least
57445 PTHREAD_DESTRUCTOR_ITERATIONS attempts at destruction) or in an infinite loop.

57446 Both functions may be implemented as macros.

57447 **RETURN VALUE**

57448 The *pthread_getspecific*( ) function shall return the thread-specific data value associated with the
57449 given *key*. If no thread-specific data value is associated with *key*, then the value NULL shall be
57450 returned.

57451 If successful, the *pthread_setspecific*( ) function shall return zero; otherwise, an error number shall
57452 be returned to indicate the error.

57453 **ERRORS**

57454 No errors are returned from *pthread_getspecific*( ).

57455 The *pthread_setspecific*( ) function shall fail if:

57456 [ENOMEM] Insufficient memory exists to associate the non-NULL value with the key.

57457 The *pthread_setspecific*( ) function shall not return an error code of [EINTR].

57458 **EXAMPLES**

57459 None.

57460 **APPLICATION USAGE**

57461 None.

57462 **RATIONALE**

57463 Performance and ease-of-use of *pthread_getspecific*( ) are critical for functions that rely on
57464 maintaining state in thread-specific data. Since no errors are required to be detected by it, and
57465 since the only error that could be detected is the use of an invalid key, the function to
57466 *pthread_getspecific*( ) has been designed to favor speed and simplicity over error reporting.

57467 If an implementation detects that the value specified by the *key* argument to *pthread_setspecific*( )
57468 does not refer to a a key value obtained from *pthread_key_create*( ) or refers to a key that has been

  
```

deleted with *pthread_key_delete*( ), it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_key_create( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_getspecific*( ) and *pthread_setspecific*( ) functions are marked as part of the Threads option.

IEEE PASC Interpretation 1003.1c #3 (Part 6) is applied, updating the DESCRIPTION.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/96 is applied, updating the ERRORS section so that the [ENOMEM] error case is changed from ``to associate the value with the key'' to ``to associate the non-NULL value with the key''.

Issue 7

Austin Group Interpretation 1003.1-2001 #063 is applied, updating the ERRORS section.

The *pthread_getspecific*( ) and *pthread_setspecific*( ) functions are moved from the Threads option to the Base.

The [EINVAL] error for a key value not obtained from *pthread_key_create*( ) or a key deleted with *pthread_key_delete*( ) is removed; this condition results in undefined behavior.

NAME

pthread_join — wait for thread termination

SYNOPSIS

```
#include <pthread.h>

int pthread_join(pthread_t thread, void **value_ptr);
```

DESCRIPTION

The *pthread_join*( ) function shall suspend execution of the calling thread until the target *thread* terminates, unless the target *thread* has already terminated. On return from a successful *pthread_join*( ) call with a non-NULL *value_ptr* argument, the value passed to *pthread_exit*( ) by the terminating thread shall be made available in the location referenced by *value_ptr*. When a *pthread_join*( ) returns successfully, the target thread has been terminated. The results of multiple simultaneous calls to *pthread_join*( ) specifying the same target thread are undefined. If the thread calling *pthread_join*( ) is canceled, then the target thread shall not be detached.

It is unspecified whether a zombie thread counts against {PTHREAD_THREADS_MAX}.

The behavior is undefined if the value specified by the *thread* argument to *pthread_join*( ) does not refer to a joinable thread.

The behavior is undefined if the value specified by the *thread* argument to *pthread_join*( ) refers to the calling thread.

If *thread* refers to a thread that was created using *thrd_create*( ) and the thread terminates, or has already terminated, by returning from its start routine, the behavior of *pthread_join*( ) is undefined. If *thread* refers to a thread that terminates, or has already terminated, by calling *thrd_exit*( ), the behavior of *pthread_join*( ) is undefined.

RETURN VALUE

If successful, the *pthread_join*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_join*( ) function may fail if:

[EDEADLK] A deadlock was detected.

The *pthread_join*( ) function shall not return an error code of [EINTR].

EXAMPLES

An example of thread creation and deletion follows:

```
typedef struct {
    int *ar;
    long n;
} subarray;

void *
incer(void *arg)
{
    long i;

    for (i = 0; i < ((subarray *)arg)->n; i++)
        ((subarray *)arg)->ar[i]++;
}

int main(void)
{
```



```
        int        ar[1000000];
        pthread_t  th1, th2;
        subarray   sb1, sb2;

        sb1.ar = &ar[0];
        sb1.n  = 500000;
        (void) pthread_create(&th1, NULL, incer, &sb1);

        sb2.ar = &ar[500000];
        sb2.n  = 500000;
        (void) pthread_create(&th2, NULL, incer, &sb2);

        (void) pthread_join(th1, NULL);
        (void) pthread_join(th2, NULL);
        return 0;
    }
```

APPLICATION USAGE

None.

RATIONALE

The *pthread_join*( ) function is a convenience that has proven useful in multi-threaded applications. It is true that a programmer could simulate this function if it were not provided by passing extra state as part of the argument to the *start_routine*( ). The terminating thread would set a flag to indicate termination and broadcast a condition that is part of that state; a joining thread would wait on that condition variable. While such a technique would allow a thread to wait on more complex conditions (for example, waiting for multiple threads to terminate), waiting on individual thread termination is considered widely useful. Also, including the *pthread_join*( ) function in no way precludes a programmer from coding such complex waits. Thus, while not a primitive, including *pthread_join*( ) in this volume of POSIX.1-2024 was considered valuable.

The *pthread_join*( ) function provides a simple mechanism allowing an application to wait for a thread to terminate. After the thread terminates, the application may then choose to clean up resources that were used by the thread. For instance, after *pthread_join*( ) returns, any application-provided stack storage could be reclaimed.

The *pthread_join*( ) or *pthread_detach*( ) function should eventually be called for every thread that is created with the *detachstate* attribute set to PTHREAD_CREATE_JOINABLE so that storage associated with the thread may be reclaimed.

The interaction between *pthread_join*( ) and cancellation is well-defined for the following reasons:

- The *pthread_join*( ) function, like all other non-async-cancel-safe functions, can only be called with deferred cancelability type.
- Cancellation cannot occur in the disabled cancelability state.

Thus, only the default cancelability state need be considered. As specified, either the *pthread_join*( ) call is canceled, or it succeeds, but not both. The difference is obvious to the application, since either a cancellation handler is run or *pthread_join*( ) returns. There are no race conditions since *pthread_join*( ) was called in the deferred cancelability state.

If an implementation detects that the value specified by the *thread* argument to *pthread_join*( ) does not refer to a joinable thread, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the *thread* argument to *pthread_join*( )

refers to the calling thread, it is recommended that the function should fail and report an [EDEADLK] error.

If an implementation detects use of a thread ID after the end of its lifetime, it is recommended that the function should fail and report an [ESRCH] error.

The *pthread_join*( ) function cannot be used to obtain the exit status of a thread that was created using *thrd_create*( ) and which terminates by returning from its start routine, or of a thread that terminates by calling *thrd_exit*( ), because such threads have an **int** exit status, instead of the **void** * that *pthread_join*( ) returns via its *value_ptr* argument.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_create( ), *thrd_create*( ), *thrd_exit*( ), *wait*( )

XBD Section 4.15.2 (on page 104), **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_join*( ) function is marked as part of the Threads option.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/97 is applied, updating the ERRORS section so that the [EINVAL] error is made optional and the words ``the implementation has detected'' are removed from it.

Issue 7

The *pthread_join*( ) function is moved from the Threads option to the Base.

Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

The [EINVAL] error for a non-joinable thread is removed; this condition results in undefined behavior.

The [EDEADLK] error for the calling thread is removed; this condition results in undefined behavior.

Issue 8

Austin Group Defect 792 is applied, changing ``a thread that has exited but remains unjoined'' to ``a zombie thread''.

Austin Group Defect 1302 is applied, updating the page to account for the addition of **<threads.h>** interfaces.

NAME

pthread_key_create — thread-specific data key creation

SYNOPSIS

```
#include <pthread.h>

int pthread_key_create(pthread_key_t *key, void (*destructor)(void*));
```

DESCRIPTION

The *pthread_key_create*( ) function shall create a thread-specific data key visible to all threads in the process. Key values provided by *pthread_key_create*( ) are opaque objects used to locate thread-specific data. Although the same key value may be used by different threads, the values bound to the key by *pthread_setspecific*( ) are maintained on a per-thread basis and persist for the life of the calling thread.

Upon key creation, the value NULL shall be associated with the new key in all active threads. Upon thread creation, the value NULL shall be associated with all defined keys in the new thread.

An optional destructor function may be associated with each key value. At thread exit, if a key value has a non-NULL destructor pointer, and the thread has a non-NULL value associated with that key, the value of the key is set to NULL, and then the function pointed to is called with the previously associated value as its sole argument. The order of destructor calls is unspecified if more than one destructor exists for a thread when it exits.

If, after all the destructors have been called for all non-NULL values with associated destructors, there are still some non-NULL values with associated destructors, then the process is repeated. If, after at least {PTHREAD_DESTRUCTOR_ITERATIONS} iterations of destructor calls for outstanding non-NULL values, there are still some non-NULL values with associated destructors, implementations may stop calling destructors, or they may continue calling destructors until no non-NULL values with associated destructors exist, even though this might result in an infinite loop.

RETURN VALUE

If successful, the *pthread_key_create*( ) function shall store the newly created key value at **key* and shall return zero. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_key_create*( ) function shall fail if:

[EAGAIN] The system lacked the necessary resources to create another thread-specific data key, or the system-imposed limit on the total number of keys per process {PTHREAD_KEYS_MAX} has been exceeded.

[ENOMEM] Insufficient memory exists to create the key.

The *pthread_key_create*( ) function shall not return an error code of [EINTR].

EXAMPLES

The following example demonstrates a function that initializes a thread-specific data key when it is first called, and associates a thread-specific object with each calling thread, initializing this object when necessary.

```
static pthread_key_t key;
static pthread_once_t key_once = PTHREAD_ONCE_INIT;

static void
make_key()
{
```



```
57657           (void) pthread_key_create(&key, NULL);
57658       }
57659       func()
57660       {
57661           void *ptr;
57662           (void) pthread_once(&key_once, make_key);
57663           if ((ptr = pthread_getspecific(key)) == NULL) {
57664               ptr = malloc(OBJECT_SIZE);
57665               ...
57666               (void) pthread_setspecific(key, ptr);
57667           }
57668           ...
57669       }
```

Note that the key has to be initialized before *pthread_getspecific*( ) or *pthread_setspecific*( ) can be used. The *pthread_key_create*( ) call could either be explicitly made in a module initialization routine, or it can be done implicitly by the first call to a module as in this example. Any attempt to use the key before it is initialized is a programming error, making the code below incorrect.

```
57674       static pthread_key_t key;
57675       func()
57676       {
57677           void *ptr;
57678          /* KEY NOT INITIALIZED!!!  THIS WILL NOT WORK!!! */
57679           if ((ptr = pthread_getspecific(key)) == NULL &&
57680               pthread_setspecific(key, NULL) != 0) {
57681               pthread_key_create(&key, NULL);
57682               ...
57683           }
57684       }
```

APPLICATION USAGE

None.

RATIONALE

Destructor Functions

Normally, the value bound to a key on behalf of a particular thread is a pointer to storage allocated dynamically on behalf of the calling thread. The destructor functions specified with *pthread_key_create*( ) are intended to be used to free this storage when the thread exits. Thread cancellation cleanup handlers cannot be used for this purpose because thread-specific data may persist outside the lexical scope in which the cancellation cleanup handlers operate.

If the value associated with a key needs to be updated during the lifetime of the thread, it may be necessary to release the storage associated with the old value before the new value is bound. Although the *pthread_setspecific*( ) function could do this automatically, this feature is not needed often enough to justify the added complexity. Instead, the programmer is responsible for freeing the stale storage:

```
57699       old = pthread_getspecific(key);
57700       new = allocate();
57701       destructor(old);
```



```
pthread_setspecific(key, new);
```

Note: The above example could leak storage if run with asynchronous cancellation enabled. No such problems occur in the default cancellation state if no cancellation points occur between the get and set.

There is no notion of a destructor-safe function. If an application does not call *pthread_exit*( ) from a signal handler, or if it blocks any signal whose handler may call *pthread_exit*( ) while calling async-unsafe functions, all functions may be safely called from destructors.

Non-Idempotent Data Key Creation

There were requests to make *pthread_key_create*( ) idempotent with respect to a given *key* address parameter. This would allow applications to call *pthread_key_create*( ) multiple times for a given *key* address and be guaranteed that only one key would be created. Doing so would require the key value to be previously initialized (possibly at compile time) to a known null value and would require that implicit mutual-exclusion be performed based on the address and contents of the *key* parameter in order to guarantee that exactly one key would be created.

Unfortunately, the implicit mutual-exclusion would not be limited to only *pthread_key_create*( ). On many implementations, implicit mutual-exclusion would also have to be performed by *pthread_getspecific*( ) and *pthread_setspecific*( ) in order to guard against using incompletely stored or not-yet-visible key values. This could significantly increase the cost of important operations, particularly *pthread_getspecific*( ).

Thus, this proposal was rejected. The *pthread_key_create*( ) function performs no implicit synchronization. It is the responsibility of the programmer to ensure that it is called exactly once per key before use of the key. Several straightforward mechanisms can already be used to accomplish this, including calling explicit module initialization functions, using mutexes, and using *pthread_once*( ). This places no significant burden on the programmer, introduces no possibly confusing *ad hoc* implicit synchronization mechanism, and potentially allows commonly used thread-specific data operations to be more efficient.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_getspecific( ), *pthread_key_delete*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_key_create*( ) function is marked as part of the Threads option.

IEEE PASC Interpretation 1003.1c #8 is applied, updating the DESCRIPTION.

Issue 7

The *pthread_key_create*( ) function is moved from the Threads option to the Base.

Issue 8

Austin Group Defect 1059 is applied, changing the example code in the RATIONALE section.

NAME

pthread_key_delete — thread-specific data key deletion

SYNOPSIS

```
#include <pthread.h>

int pthread_key_delete(pthread_key_t key);
```

DESCRIPTION

The *pthread_key_delete*( ) function shall delete a thread-specific data key previously returned by *pthread_key_create*( ). The thread-specific data values associated with *key* need not be NULL at the time *pthread_key_delete*( ) is called. It is the responsibility of the application to free any application storage or perform any cleanup actions for data structures related to the deleted key or associated thread-specific data in any threads; this cleanup can be done either before or after *pthread_key_delete*( ) is called. Any attempt to use *key* following the call to *pthread_key_delete*( ) results in undefined behavior.

The *pthread_key_delete*( ) function shall be callable from within destructor functions. No destructor functions shall be invoked by *pthread_key_delete*( ). Any destructor function that may have been associated with *key* shall no longer be called upon thread exit.

RETURN VALUE

If successful, the *pthread_key_delete*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_key_delete*( ) function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

A thread-specific data key deletion function has been included in order to allow the resources associated with an unused thread-specific data key to be freed. Unused thread-specific data keys can arise, among other scenarios, when a dynamically loaded module that allocated a key is unloaded.

Conforming applications are responsible for performing any cleanup actions needed for data structures associated with the key to be deleted, including data referenced by thread-specific data values. No such cleanup is done by *pthread_key_delete*( ). In particular, destructor functions are not called. There are several reasons for this division of responsibility:

1. The associated destructor functions used to free thread-specific data at thread exit time are only guaranteed to work correctly when called in the thread that allocated the thread-specific data. (Destructors themselves may utilize thread-specific data.) Thus, they cannot be used to free thread-specific data in other threads at key deletion time. Attempting to have them called by other threads at key deletion time would require other threads to be asynchronously interrupted. But since interrupted threads could be in an arbitrary state, including holding locks necessary for the destructor to run, this approach would fail. In general, there is no safe mechanism whereby an implementation could free thread-specific data at key deletion time.

2. Even if there were a means of safely freeing thread-specific data associated with keys to be deleted, doing so would require that implementations be able to enumerate the threads with non-NULL data and potentially keep them from creating more thread-

specific data while the key deletion is occurring. This special case could cause extra synchronization in the normal case, which would otherwise be unnecessary.

For an application to know that it is safe to delete a key, it has to know that all the threads that might potentially ever use the key do not attempt to use it again. For example, it could know this if all the client threads have called a cleanup procedure declaring that they are through with the module that is being shut down, perhaps by setting a reference count to zero.

If an implementation detects that the value specified by the *key* argument to *pthread_key_delete*( ) does not refer to a a key value obtained from *pthread_key_create*( ) or refers to a key that has been deleted with *pthread_key_delete*( ), it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_key_create( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_key_delete*( ) function is marked as part of the Threads option.

Issue 7

The *pthread_key_delete*( ) function is moved from the Threads option to the Base.

The [EINVAL] error for a key value not obtained from *pthread_key_create*( ) or a key deleted with *pthread_key_delete*( ) is removed; this condition results in undefined behavior.

NAME

pthread_kill — send a signal to a thread

SYNOPSIS

CX
```
#include <signal.h>

int pthread_kill(pthread_t thread, int sig);
```

DESCRIPTION

The *pthread_kill*( ) function shall request that a signal be delivered to the specified thread. It shall not be an error if *thread* is a zombie thread.

RETURN VALUE

Upon successful completion, the function shall return a value of zero. Otherwise, the function shall return an error number. If the *pthread_kill*( ) function fails, no signal shall be sent.

ERRORS

The *pthread_kill*( ) function may fail if:

[EINVAL] The value of the *sig* argument is zero.

The *pthread_kill*( ) function shall fail if:

[EINVAL] The value of the *sig* argument is non-zero and is an invalid or unsupported signal number.

The *pthread_kill*( ) function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

The *pthread_kill*( ) function provides a mechanism for asynchronously directing a signal at a thread in the calling process. This could be used, for example, by one thread to affect broadcast delivery of a signal to a set of threads.

Note that *pthread_kill*( ) only causes the signal to be handled in the context of the given thread; the signal action (termination or stopping) affects the process as a whole.

RATIONALE

If an implementation detects use of a thread ID after the end of its lifetime, it is recommended that the function should fail and report an [ESRCH] error.

Historical implementations varied on the result of a *pthread_kill*( ) with a thread ID indicating a zombie thread. Some indicated success on such a call, while others gave an error of [ESRCH]. Since the definition of thread lifetime in this volume of POSIX.1-2024 covers zombie threads, the [ESRCH] error as described is inappropriate in this case and implementations that give this error do not conform. In particular, this means that an application cannot have one thread check for termination of another by calling *pthread_kill*( ) with a *sig* argument of zero, and implementations may indicate that it is not possible by returning [EINVAL] when *sig* is zero.

FUTURE DIRECTIONS

None.

SEE ALSO

kill( ), *pthread_self*( ), *raise*( )

XBD **<signal.h>**

57852 **CHANGE HISTORY**

57853 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

57854 **Issue 6**

57855 The *pthread_kill*( ) function is marked as part of the Threads option.

57856 The APPLICATION USAGE section is added.

57857 **Issue 7**

57858 The *pthread_kill*( ) function is moved from the Threads option to the Base.

57859 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

57860 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0277 [765] is applied.

57861 **Issue 8**

57862 Austin Group Defect 792 is applied, adding a requirement that passing the thread ID of a zombie
57863 thread to *pthread_kill*( ) is not treated as an error.

57864 Austin Group Defect 1214 is applied, allowing *pthread_kill*( ) to fail with [EINVAL] when the *sig*
57865 argument is zero.

NAME

pthread_mutex_clocklock, pthread_mutex_timedlock — lock a mutex

SYNOPSIS

```
#include <pthread.h>

int pthread_mutex_clocklock(pthread_mutex_t *restrict mutex,
    clockid_t clock_id, const struct timespec *restrict abstime);
int pthread_mutex_timedlock(pthread_mutex_t *restrict mutex,
    const struct timespec *restrict abstime);
```

DESCRIPTION

The *pthread_mutex_clocklock*( ) and *pthread_mutex_timedlock*( ) functions shall lock the mutex object referenced by *mutex*. If the mutex is already locked, the calling thread shall block until the mutex becomes available as in the *pthread_mutex_lock*( ) function. If the mutex cannot be locked without waiting for another thread to unlock the mutex, this wait shall be terminated when the specified timeout expires.

The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the call.

For *pthread_mutex_timedlock*( ), the timeout shall be based on the CLOCK_REALTIME clock. For *pthread_mutex_clocklock*( ), the timeout shall be based on the clock specified by the *clock_id* argument. The resolution of the timeout shall be the resolution of the clock on which it is based. Implementations shall support passing CLOCK_REALTIME and CLOCK_MONOTONIC to *pthread_mutex_clocklock*( ) as the *clock_id* argument.

Under no circumstance shall the function fail with a timeout if the mutex can be locked immediately. The validity of the *abstime* parameter need not be checked if the mutex can be locked immediately.

RPI | TPI As a consequence of the priority inheritance rules (for mutexes initialized with the PRIO_INHERIT protocol), if a timed mutex wait is terminated because its timeout expires, the priority of the owner of the mutex shall be adjusted as necessary to reflect the fact that this thread is no longer among the threads waiting for the mutex.

If *mutex* is a robust mutex and the process containing the owning thread terminated while holding the mutex lock, a call to *pthread_mutex_clocklock*( ) or *pthread_mutex_timedlock*( ) shall return the error value [EOWNERDEAD]. If *mutex* is a robust mutex and the owning thread terminated while holding the mutex lock, a call to *pthread_mutex_clocklock*( ) or *pthread_mutex_timedlock*( ) may return the error value [EOWNERDEAD] even if the process in which the owning thread resides has not terminated. In these cases, the mutex is locked by the thread but the state it protects is marked as inconsistent. The application should ensure that the state is made consistent for reuse and when that is complete call *pthread_mutex_consistent*( ). If the application is unable to recover the state, it should unlock the mutex without a prior call to *pthread_mutex_consistent*( ), after which the mutex is marked permanently unusable.

If *mutex* does not refer to an initialized mutex object, the behavior is undefined.

RETURN VALUE

If successful, the *pthread_mutex_clocklock*( ) and *pthread_mutex_timedlock*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutex_clocklock*( ) and *pthread_mutex_timedlock*( ) functions shall fail if:

[EAGAIN] The mutex could not be acquired because the maximum number of recursive locks for *mutex* has been exceeded.

[EAGAIN] The mutex is a robust mutex and the system resources available for robust mutexes owned would be exceeded.

[EDEADLK] The mutex type is PTHREAD_MUTEX_ERRORCHECK and the current thread already owns the mutex.

[EINVAL] The mutex was created with the protocol attribute having the value PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than the mutex' current priority ceiling.

[EINVAL] The process or thread would have blocked, and either the *abstime* parameter specified a nanoseconds field value less than zero or greater than or equal to 1 000 million, or the *pthread_mutex_clocklock*( ) function was passed an invalid or unsupported *clock_id* value.

[ENOTRECOVERABLE]
The state protected by the mutex is not recoverable.

[EOWNERDEAD]
The mutex is a robust mutex and the process containing the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent.

[ETIMEDOUT] The mutex could not be locked before the specified timeout expired.

The *pthread_mutex_clocklock*( ) and *pthread_mutex_timedlock*( ) functions may fail if:

[EDEADLK] A deadlock condition was detected.

[EOWNERDEAD]
The mutex is a robust mutex and the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications that have assumed that non-zero return values are errors will need updating for use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If an application is supposed to work with normal and robust mutexes, it should check all return values for error conditions and if necessary take appropriate action.

RATIONALE

Refer to *pthread_mutex_lock*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_mutex_destroy( ), *pthread_mutex_lock*( ), *time*( )

XBD Section 4.15.2 (on page 104), **<pthread.h>**, **<time.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/99 is applied, marking the last paragraph in the DESCRIPTION as part of the Thread Priority Inheritance option.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/100 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

Issue 7

Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

The *pthread_mutex_timedlock*( ) function is moved from the Timeouts option to the Base.

Functionality relating to the Timers option is moved to the Base.

The DESCRIPTION is updated to clarify the behavior when *mutex* does not refer to an initialized mutex.

The ERRORS section is updated to account properly for all of the various mutex types.

Issue 8

Austin Group Defect 354 is applied, adding the [EAGAIN] error for exceeding system resources available for robust mutexes owned.

Austin Group Defect 592 is applied, removing text relating to **<time.h>** from the SYNOPSIS and DESCRIPTION sections.

Austin Group Defects 1216 and 1472 are applied, adding *pthread_mutex_clocklock*( ).

57975 **NAME**

57976 pthread_mutex_consistent — mark state protected by robust mutex as consistent

57977 **SYNOPSIS**

57978 ```
#include <pthread.h>
```

57979 ```
int pthread_mutex_consistent(pthread_mutex_t *mutex);
```

57980 **DESCRIPTION**

57981 If *mutex* is a robust mutex in an inconsistent state, the *pthread_mutex_consistent*( ) function can be
57982 used to mark the state protected by the mutex referenced by *mutex* as consistent again.

57983 If an owner of a robust mutex terminates while holding the mutex, the mutex becomes
57984 inconsistent and the next thread that acquires the mutex lock shall be notified of the state by the
57985 return value [EOWNERDEAD]. In this case, the mutex does not become normally usable again
57986 until the state is marked consistent.

57987 If the thread which acquired the mutex lock with the return value [EOWNERDEAD] terminates
57988 before calling either *pthread_mutex_consistent*( ) or *pthread_mutex_unlock*( ), the next thread that
57989 acquires the mutex lock shall be notified about the state of the mutex by the return value
57990 [EOWNERDEAD].

57991 The behavior is undefined if the value specified by the *mutex* argument to
57992 *pthread_mutex_consistent*( ) does not refer to an initialized mutex.

57993 **RETURN VALUE**

57994 Upon successful completion, the *pthread_mutex_consistent*( ) function shall return zero.
57995 Otherwise, an error value shall be returned to indicate the error.

57996 **ERRORS**

57997 The *pthread_mutex_consistent*( ) function shall fail if:

57998 [EINVAL] The mutex object referenced by *mutex* is not robust or does not protect an
57999 inconsistent state.

58000 These functions shall not return an error code of [EINTR].

58001 **EXAMPLES**

58002 None.

58003 **APPLICATION USAGE**

58004 The *pthread_mutex_consistent*( ) function is only responsible for notifying the implementation that
58005 the state protected by the mutex has been recovered and that normal operations with the mutex
58006 can be resumed. It is the responsibility of the application to recover the state so it can be reused.
58007 If the application is not able to perform the recovery, it can notify the implementation that the
58008 situation is unrecoverable by a call to *pthread_mutex_unlock*( ) without a prior call to
58009 *pthread_mutex_consistent*( ), in which case subsequent threads that attempt to lock the mutex will
58010 fail to acquire the lock and be returned [ENOTRECOVERABLE].

58011 **RATIONALE**

58012 If an implementation detects that the value specified by the *mutex* argument to
58013 *pthread_mutex_consistent*( ) does not refer to an initialized mutex, it is recommended that the
58014 function should fail and report an [EINVAL] error.

58015 **FUTURE DIRECTIONS**

58016 None.

58017 **SEE ALSO**

58018 *pthread_mutex_lock( )*, *pthread_mutexattr_getrobust( )*

58019 XBD **<pthread.h>**

58020 **CHANGE HISTORY**

58021 First released in Issue 7.

58022 **NAME**

58023 pthread_mutex_destroy, pthread_mutex_init — destroy and initialize a mutex

58024 **SYNOPSIS**

58025 ```
#include <pthread.h>

int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
    const pthread_mutexattr_t *restrict attr);
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

58030 **DESCRIPTION**

58031 The *pthread_mutex_destroy*( ) function shall destroy the mutex object referenced by *mutex*; the
58032 mutex object becomes, in effect, uninitialized. An implementation may cause
58033 *pthread_mutex_destroy*( ) to set the object referenced by *mutex* to an invalid value.

58034 A destroyed mutex object can be reinitialized using *pthread_mutex_init*( ); the results of otherwise
58035 referencing the object after it has been destroyed are undefined.

58036 It shall be safe to destroy an initialized mutex that is unlocked. Attempting to destroy a locked
58037 mutex, or a mutex that another thread is attempting to lock, or a mutex that is being used in a
58038 *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), or *pthread_cond_wait*( ) call by another thread,
58039 results in undefined behavior.

58040 The *pthread_mutex_init*( ) function shall initialize the mutex referenced by *mutex* with attributes
58041 specified by *attr*. If *attr* is NULL, the default mutex attributes are used; the effect shall be the
58042 same as passing the address of a default mutex attributes object. Upon successful initialization,
58043 the state of the mutex becomes initialized and unlocked.

58044 See Section 2.9.9 (on page 548) for further requirements.

58045 Attempting to initialize an already initialized mutex results in undefined behavior.

58046 In cases where default mutex attributes are appropriate, the macro
58047 PTHREAD_MUTEX_INITIALIZER can be used to initialize mutexes. The effect shall be
58048 equivalent to dynamic initialization by a call to *pthread_mutex_init*( ) with parameter *attr*
58049 specified as NULL, except that no error checks are performed.

58050 The behavior is undefined if the value specified by the *mutex* argument to
58051 *pthread_mutex_destroy*( ) does not refer to an initialized mutex.

58052 The behavior is undefined if the value specified by the *attr* argument to *pthread_mutex_init*( )
58053 does not refer to an initialized mutex attributes object.

58054 **RETURN VALUE**

58055 If successful, the *pthread_mutex_destroy*( ) and *pthread_mutex_init*( ) functions shall return zero;
58056 otherwise, an error number shall be returned to indicate the error.

58057 **ERRORS**

58058 The *pthread_mutex_init*( ) function shall fail if:

58059 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize
58060 another mutex.

58061 [ENOMEM] Insufficient memory exists to initialize the mutex.

58062 [EPERM] The caller does not have the privilege to perform the operation.

  
```

The *pthread_mutex_init*( ) function may fail if:

[EINVAL] The attributes object referenced by *attr* has the robust mutex attribute set without the process-shared attribute being set.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *mutex* argument to *pthread_mutex_destroy*( ) does not refer to an initialized mutex, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the *mutex* argument to *pthread_mutex_destroy*( ) or *pthread_mutex_init*( ) refers to a locked mutex or a mutex that is referenced (for example, while being used in a *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), or *pthread_cond_wait*( ) call) by another thread, or detects that the value specified by the *mutex* argument to *pthread_mutex_init*( ) refers to an already initialized mutex, it is recommended that the function should fail and report an [EBUSY] error.

If an implementation detects that the value specified by the *attr* argument to *pthread_mutex_init*( ) does not refer to an initialized mutex attributes object, it is recommended that the function should fail and report an [EINVAL] error.

Alternate Implementations Possible

This volume of POSIX.1-2024 supports several alternative implementations of mutexes. An implementation may store the lock directly in the object of type **pthread_mutex_t**. Alternatively, an implementation may store the lock in the heap and merely store a pointer, handle, or unique ID in the mutex object. Either implementation has advantages or may be required on certain hardware configurations. So that portable code can be written that is invariant to this choice, this volume of POSIX.1-2024 does not define assignment or equality for this type, and it uses the term ``initialize'' to reinforce the (more restrictive) notion that the lock may actually reside in the mutex object itself.

Note that this precludes an over-specification of the type of the mutex or condition variable and motivates the opaqueness of the type.

An implementation is permitted, but not required, to have *pthread_mutex_destroy*( ) store an illegal value into the mutex. This may help detect erroneous programs that try to lock (or otherwise reference) a mutex that has already been destroyed.

Tradeoff Between Error Checks and Performance Supported

Many error conditions that can occur are not required to be detected by the implementation in order to let implementations trade off performance *versus* degree of error checking according to the needs of their specific applications and execution environment. As a general rule, conditions caused by the system (such as insufficient memory) are required to be detected, but conditions caused by an erroneously coded application (such as failing to provide adequate synchronization to prevent a mutex from being deleted while in use) are specified to result in undefined behavior.

A wide range of implementations is thus made possible. For example, an implementation

intended for application debugging may implement all of the error checks, but an implementation running a single, provably correct application under very tight performance constraints in an embedded computer might implement minimal checks. An implementation might even be provided in two versions, similar to the options that compilers provide: a full-checking, but slower version; and a limited-checking, but faster version. To forbid this optionality would be a disservice to users.

By carefully limiting the use of ``undefined behavior'' only to things that an erroneous (badly coded) application might do, and by defining that resource-not-available errors are mandatory, this volume of POSIX.1-2024 ensures that a fully-conforming application is portable across the full range of implementations, while not forcing all implementations to add overhead to check for numerous things that a correct program never does. When the behavior is undefined, no error number is specified to be returned on implementations that do detect the condition. This is because undefined behavior means *anything* can happen, which includes returning with any value (which might happen to be a valid, but different, error number). However, since the error number might be useful to application developers when diagnosing problems during application development, a recommendation is made in rationale that implementors should return a particular error number if their implementation does detect the condition.

Why No Limits are Defined

Defining symbols for the maximum number of mutexes and condition variables was considered but rejected because the number of these objects may change dynamically. Furthermore, many implementations place these objects into application memory; thus, there is no explicit maximum.

Static Initializers for Mutexes and Condition Variables

Providing for static initialization of statically allocated synchronization objects allows modules with private static synchronization variables to avoid runtime initialization tests and overhead. Furthermore, it simplifies the coding of self-initializing modules. Such modules are common in C libraries, where for various reasons the design calls for self-initialization instead of requiring an explicit module initialization function to be called. An example use of static initialization follows.

Without static initialization, a self-initializing routine *foo*( ) might look as follows:

```
static pthread_once_t foo_once = PTHREAD_ONCE_INIT;
static pthread_mutex_t foo_mutex;

void foo_init()
{
    pthread_mutex_init(&foo_mutex, NULL);
}

void foo()
{
    pthread_once(&foo_once, foo_init);
    pthread_mutex_lock(&foo_mutex);
   /* Do work. */
    pthread_mutex_unlock(&foo_mutex);
}
```

With static initialization, the same routine could be coded as follows:

```
static pthread_mutex_t foo_mutex = PTHREAD_MUTEX_INITIALIZER;
```



```
void foo()
{
    pthread_mutex_lock(&foo_mutex);
   /* Do work. */
    pthread_mutex_unlock(&foo_mutex);
}
```

Note that the static initialization both eliminates the need for the initialization test inside *pthread_once*( ) and the fetch of *&foo_mutex* to learn the address to be passed to *pthread_mutex_lock*( ) or *pthread_mutex_unlock*( ).

Thus, the C code written to initialize static objects is simpler on all systems and is also faster on a large class of systems; those where the (entire) synchronization object can be stored in application memory.

Yet the locking performance question is likely to be raised for machines that require mutexes to be allocated out of special memory. Such machines actually have to have mutexes and possibly condition variables contain pointers to the actual hardware locks. For static initialization to work on such machines, *pthread_mutex_lock*( ) also has to test whether or not the pointer to the actual lock has been allocated. If it has not, *pthread_mutex_lock*( ) has to initialize it before use. The reservation of such resources can be made when the program is loaded, and hence return codes have not been added to mutex locking and condition variable waiting to indicate failure to complete initialization.

This runtime test in *pthread_mutex_lock*( ) would at first seem to be extra work; an extra test is required to see whether the pointer has been initialized. On most machines this would actually be implemented as a fetch of the pointer, testing the pointer against zero, and then using the pointer if it has already been initialized. While the test might seem to add extra work, the extra effort of testing a register is usually negligible since no extra memory references are actually done. As more and more machines provide caches, the real expenses are memory references, not instructions executed.

Alternatively, depending on the machine architecture, there are often ways to eliminate *all* overhead in the most important case: on the lock operations that occur *after* the lock has been initialized. This can be done by shifting more overhead to the less frequent operation: initialization. Since out-of-line mutex allocation also means that an address has to be dereferenced to find the actual lock, one technique that is widely applicable is to have static initialization store a bogus value for that address; in particular, an address that causes a machine fault to occur. When such a fault occurs upon the first attempt to lock such a mutex, validity checks can be done, and then the correct address for the actual lock can be filled in. Subsequent lock operations incur no extra overhead since they do not ``fault''. This is merely one technique that can be used to support static initialization, while not adversely affecting the performance of lock acquisition. No doubt there are other techniques that are highly machine-dependent.

The locking overhead for machines doing out-of-line mutex allocation is thus similar for modules being implicitly initialized, where it is improved for those doing mutex allocation entirely inline. The inline case is thus made much faster, and the out-of-line case is not significantly worse.

Besides the issue of locking performance for such machines, a concern is raised that it is possible that threads would serialize contending for initialization locks when attempting to finish initializing statically allocated mutexes. (Such finishing would typically involve taking an internal lock, allocating a structure, storing a pointer to the structure in the mutex, and releasing the internal lock.) First, many implementations would reduce such serialization by hashing on the mutex address. Second, such serialization can only occur a bounded number of times. In

particular, it can happen at most as many times as there are statically allocated synchronization objects. Dynamically allocated objects would still be initialized via *pthread_mutex_init*( ) or *pthread_cond_init*( ).

Finally, if none of the above optimization techniques for out-of-line allocation yields sufficient performance for an application on some implementation, the application can avoid static initialization altogether by explicitly initializing all synchronization objects with the corresponding *pthread_*_init*( ) functions, which are supported by all implementations. An implementation can also document the tradeoffs and advise which initialization technique is more efficient for that particular implementation.

Destroying Mutexes

A mutex can be destroyed immediately after it is unlocked. However, since attempting to destroy a locked mutex, or a mutex that another thread is attempting to lock, or a mutex that is being used in a *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), or *pthread_cond_wait*( ) call by another thread, results in undefined behavior, care must be taken to ensure that no other thread may be referencing the mutex.

Robust Mutexes

Implementations are required to provide robust mutexes for mutexes with the process-shared attribute set to PTHREAD_PROCESS_SHARED. Implementations are allowed, but not required, to provide robust mutexes when the process-shared attribute is set to PTHREAD_PROCESS_PRIVATE.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_mutex_getprioceiling( ), *pthread_mutexattr_getrobust*( ), *pthread_mutex_clocklock*( ), *pthread_mutex_lock*( ), *pthread_mutexattr_getpshared*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_mutex_destroy*( ) and *pthread_mutex_init*( ) functions are marked as part of the Threads option.

The *pthread_mutex_timedlock*( ) function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

The **restrict** keyword is added to the *pthread_mutex_init*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

The *pthread_mutex_destroy*( ) and *pthread_mutex_init*( ) functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized mutex or an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

The [EBUSY] error for a locked mutex, a mutex that is referenced, or an already initialized mutex is removed; this condition results in undefined behavior.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0460 [70,428] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0278 [811], XSH/TC2-2008/0279 [972], and XSH/TC2-2008/0280 [811] are applied.

Issue 8

Austin Group Defect 1216 is applied, adding *pthread_cond_clockwait*( ).

NAME

pthread_mutex_getprioceiling, pthread_mutex_setprioceiling — get and set the priority ceiling of a mutex (**REALTIME THREADS**)

SYNOPSIS

RPP|TPP

```
#include <pthread.h>

int pthread_mutex_getprioceiling(const pthread_mutex_t *restrict mutex,
    int *restrict prioceiling);
int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
    int prioceiling, int *restrict old_ceiling);
```

DESCRIPTION

The *pthread_mutex_getprioceiling*( ) function shall return the current priority ceiling of the mutex.

The *pthread_mutex_setprioceiling*( ) function shall attempt to lock the mutex as if by a call to *pthread_mutex_lock*( ), except that the process of locking the mutex need not adhere to the priority protect protocol. On acquiring the mutex it shall change the mutex's priority ceiling and then release the mutex as if by a call to *pthread_mutex_unlock*( ). When the change is successful, the previous value of the priority ceiling shall be returned in *old_ceiling*.

If the *pthread_mutex_setprioceiling*( ) function fails, the mutex priority ceiling shall not be changed.

RETURN VALUE

If successful, the *pthread_mutex_getprioceiling*( ) and *pthread_mutex_setprioceiling*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions shall fail if:

[EINVAL] The protocol attribute of *mutex* is PTHREAD_PRIO_NONE.

[EPERM] The implementation requires appropriate privileges to perform the operation and the caller does not have appropriate privileges.

The *pthread_mutex_setprioceiling*( ) function shall fail if:

[EAGAIN] The mutex could not be acquired because the maximum number of recursive locks for *mutex* has been exceeded.

[EAGAIN] The mutex is a robust mutex and the system resources available for robust mutexes owned would be exceeded.

[EDEADLK] The mutex type is PTHREAD_MUTEX_ERRORCHECK and the current thread already owns the mutex.

[EINVAL] The mutex was created with the protocol attribute having the value PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than the mutex's current priority ceiling, and the implementation adheres to the priority protect protocol in the process of locking the mutex.

[ENOTRECOVERABLE]
The mutex is a robust mutex and the state protected by the mutex is not recoverable.

[EOWNERDEAD]
The mutex is a robust mutex and the process containing the previous owning thread terminated while holding the mutex lock. The mutex lock shall be

acquired by the calling thread and it is up to the new owner to make the state consistent (see *pthread_mutex_lock*( )).

The *pthread_mutex_setprioceiling*( ) function may fail if:

[EDEADLK] A deadlock condition was detected.

[EINVAL] The priority requested by *prioceiling* is out of range.

[EOWNERDEAD]
The mutex is a robust mutex and the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent (see *pthread_mutex_lock*( )).

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_mutex_clocklock( ), *pthread_mutex_destroy*( ), *pthread_mutex_lock*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

Issue 6

The *pthread_mutex_getprioceiling*( ) and *pthread_mutex_setprioceiling*( ) functions are marked as part of the Threads and Thread Priority Protection options.

The [ENOSYS] error conditions have been removed.

The *pthread_mutex_timedlock*( ) function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *pthread_mutex_getprioceiling*( ) and *pthread_mutex_setprioceiling*( ) prototypes for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

SD5-XSH-ERN-39 is applied.

Austin Group Interpretation 1003.1-2001 #052 is applied, adding [EDEADLK] as a ``may fail'' error.

SD5-XSH-ERN-158 is applied, updating the ERRORS section to include a ``shall fail'' error case for when the protocol attribute of *mutex* is PTHREAD_PRIO_NONE.

The *pthread_mutex_getprioceiling*( ) and *pthread_mutex_setprioceiling*( ) functions are moved from the Threads option to require support of either the Robust Mutex Priority Protection option or

the Non-Robust Mutex Priority Protection option.

The DESCRIPTION and ERRORS sections are updated to account properly for all of the various mutex types.

Issue 8

Austin Group Defect 354 is applied, adding the [EAGAIN] error for exceeding system resources available for robust mutexes owned.

58339 **NAME**

58340 pthread_mutex_init — destroy and initialize a mutex

58341 **SYNOPSIS**

58342 ```
#include <pthread.h>
```

58343 ```
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
```

58344 ```
    const pthread_mutexattr_t *restrict attr);
```

58345 ```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

58346 **DESCRIPTION**

58347 Refer to *pthread_mutex_destroy( )*.

NAME

pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock — lock and unlock a mutex

SYNOPSIS

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

DESCRIPTION

The mutex object referenced by *mutex* shall be locked by a call to *pthread_mutex_lock*( ) that returns zero or [EOWNERDEAD]. If the mutex is already locked by another thread, the calling thread shall block until the mutex becomes available. This operation shall return with the mutex object referenced by *mutex* in the locked state with the calling thread as its owner. If a thread attempts to relock a mutex that it has already locked, *pthread_mutex_lock*( ) shall behave as described in the **Relock** column of the following table. If a thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, *pthread_mutex_unlock*( ) shall behave as described in the **Unlock When Not Owner** column of the following table.

Mutex Type	Robustness	Relock	Unlock When Not Owner
NORMAL	non-robust	deadlock	undefined behavior
NORMAL	robust	deadlock	error returned
ERRORCHECK	either	error returned	error returned
RECURSIVE	either	recursive (see below)	error returned
DEFAULT	non-robust	undefined behavior†	undefined behavior†
DEFAULT	robust	undefined behavior†	error returned

† If the mutex type is PTHREAD_MUTEX_DEFAULT, the behavior of *pthread_mutex_lock*( ) may correspond to one of the three other standard mutex types as described in the table above. If it does not correspond to one of those three, the behavior is undefined for the cases marked †.

Where the table indicates recursive behavior, the mutex shall maintain the concept of a lock count. When a thread successfully acquires a mutex for the first time, the lock count shall be set to one. Every time a thread relocks this mutex, the lock count shall be incremented by one. Each time the thread unlocks the mutex, the lock count shall be decremented by one. When the lock count reaches zero, the mutex shall become available for other threads to acquire.

The *pthread_mutex_trylock*( ) function shall be equivalent to *pthread_mutex_lock*( ), except that if the mutex object referenced by *mutex* is currently locked (by any thread, including the current thread), the call shall return immediately. If the mutex type is PTHREAD_MUTEX_RECURSIVE and the mutex is currently owned by the calling thread, the mutex lock count shall be incremented by one and the *pthread_mutex_trylock*( ) function shall immediately return success.

The *pthread_mutex_unlock*( ) function shall release the mutex object referenced by *mutex*. The manner in which a mutex is released is dependent upon the mutex's type attribute. If there are threads blocked on the mutex object referenced by *mutex* when *pthread_mutex_unlock*( ) is called, resulting in the mutex becoming available, the scheduling policy shall determine which thread shall acquire the mutex.

(In the case of PTHREAD_MUTEX_RECURSIVE mutexes, the mutex shall become available when the count reaches zero and the calling thread no longer has any locks on this mutex.)

If a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the thread shall resume waiting for the mutex as if it was not interrupted.

If *mutex* is a robust mutex and the process containing the owning thread terminated while holding the mutex lock, a call to *pthread_mutex_lock*( ) shall return the error value [EOWNERDEAD]. If *mutex* is a robust mutex and the owning thread terminated while holding the mutex lock, a call to *pthread_mutex_lock*( ) may return the error value [EOWNERDEAD] even if the process in which the owning thread resides has not terminated. In these cases, the mutex shall be locked by the calling thread but the state it protects is marked as inconsistent. The application should ensure that the state is made consistent for reuse and when that is complete call *pthread_mutex_consistent*( ). If the application is unable to recover the state, it should unlock the mutex without a prior call to *pthread_mutex_consistent*( ), after which the mutex is marked permanently unusable.

If *mutex* does not refer to an initialized mutex object, the behavior of *pthread_mutex_lock*( ), *pthread_mutex_trylock*( ), and *pthread_mutex_unlock*( ) is undefined.

RETURN VALUE

If successful, the *pthread_mutex_lock*( ), *pthread_mutex_trylock*( ), and *pthread_mutex_unlock*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutex_lock*( ) and *pthread_mutex_trylock*( ) functions shall fail if:

[EAGAIN] The mutex could not be acquired because the maximum number of recursive locks for *mutex* has been exceeded.

[EAGAIN] The mutex is a robust mutex and the system resources available for robust mutexes owned would be exceeded.

RPP | TPP [EINVAL] The *mutex* was created with the protocol attribute having the value PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than the mutex's current priority ceiling.

[ENOTRECOVERABLE] The state protected by the mutex is not recoverable.

[EOWNERDEAD] The mutex is a robust mutex and the process containing the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent.

The *pthread_mutex_lock*( ) function shall fail if:

[EDEADLK] The mutex type is PTHREAD_MUTEX_ERRORCHECK and the current thread already owns the mutex.

The *pthread_mutex_trylock*( ) function shall fail if:

[EBUSY] The *mutex* could not be acquired because it was already locked.

The *pthread_mutex_unlock*( ) function shall fail if:

[EPERM] The mutex type is PTHREAD_MUTEX_ERRORCHECK or PTHREAD_MUTEX_RECURSIVE, or the mutex is a robust mutex, and the current thread does not own the mutex.

The *pthread_mutex_lock*( ) and *pthread_mutex_trylock*( ) functions may fail if:

[EOWNERDEAD]
The mutex is a robust mutex and the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent.

The *pthread_mutex_lock*( ) function may fail if:

[EDEADLK] A deadlock condition was detected.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications that have assumed that non-zero return values are errors will need updating for use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If an application is supposed to work with normal and robust mutexes it should check all return values for error conditions and if necessary take appropriate action.

RATIONALE

Mutex objects are intended to serve as a low-level primitive from which other thread synchronization functions can be built. As such, the implementation of mutexes should be as efficient as possible, and this has ramifications on the features available at the interface.

The mutex functions and the particular default settings of the mutex attributes have been motivated by the desire to not preclude fast, inlined implementations of mutex locking and unlocking.

Since most attributes only need to be checked when a thread is going to be blocked, the use of attributes does not slow the (common) mutex-locking case.

Likewise, while being able to extract the thread ID of the owner of a mutex might be desirable, it would require storing the current thread ID when each mutex is locked, and this could incur unacceptable levels of overhead. Similar arguments apply to a *mutex_tryunlock* operation.

For further rationale on the extended mutex types, see XRAT Threads Extensions (on page 3815).

If an implementation detects that the value specified by the *mutex* argument does not refer to an initialized mutex object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_mutex_clocklock( ), *pthread_mutex_consistent*( ), *pthread_mutex_destroy*( ), *pthread_mutexattr_getrobust*( )

XBD Section 4.15.2 (on page 104), **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_mutex_lock*( ), *pthread_mutex_trylock*( ), and *pthread_mutex_unlock*( ) functions are marked as part of the Threads option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The behavior when attempting to relock a mutex is defined.

The *pthread_mutex_timedlock*( ) function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/98 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition. The RATIONALE section is also reworded to take into account non-XSI-conformant systems.

Issue 7

SD5-XSH-ERN-43 is applied, marking the ``shall fail'' case of the [EINVAL] error as dependent on the Thread Priority Protection option.

Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

The *pthread_mutex_lock*( ), *pthread_mutex_trylock*( ), and *pthread_mutex_unlock*( ) functions are moved from the Threads option to the Base.

The following extended mutex types are moved from the XSI option to the Base:

PTHREAD_MUTEX_NORMAL
PTHREAD_MUTEX_ERRORCHECK
PTHREAD_MUTEX_RECURSIVE
PTHREAD_MUTEX_DEFAULT

The DESCRIPTION is updated to clarify the behavior when *mutex* does not refer to an initialized mutex.

The ERRORS section is updated to account properly for all of the various mutex types.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0461 [121], XSH/TC1-2008/0462 [92,428], and XSH/TC1-2008/0463 [121] are applied.

Issue 8

Austin Group Defect 354 is applied, adding the [EAGAIN] error for exceeding system resources available for robust mutexes owned.

Austin Group Defect 1115 is applied, changing ``the thread'' to ``the calling thread''.

NAME

pthread_mutex_setprioceiling — change the priority ceiling of a mutex (**REALTIME THREADS**)

SYNOPSIS

RPP|TPP

```
#include <pthread.h>

int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
    int prioceiling, int *restrict old_ceiling);
```

DESCRIPTION

Refer to *pthread_mutex_getprioceiling( )*.

58519 **NAME**

58520 pthread_mutex_timedlock — lock a mutex

58521 **SYNOPSIS**

58522
```
#include <pthread.h>
```

58523
```
int pthread_mutex_timedlock(pthread_mutex_t *restrict mutex,
```
58524
```
    const struct timespec *restrict abstime);
```

58525 **DESCRIPTION**

58526 Refer to *pthread_mutex_clocklock( )*.

58527 **NAME**

58528 pthread_mutex_trylock, pthread_mutex_unlock — lock and unlock a mutex

58529 **SYNOPSIS**

58530 `#include <pthread.h>`

58531 `int pthread_mutex_trylock(pthread_mutex_t *mutex);`
58532 `int pthread_mutex_unlock(pthread_mutex_t *mutex);`

58533 **DESCRIPTION**

58534 Refer to *pthread_mutex_lock*( ).

58535 **NAME**

58536 pthread_mutexattr_destroy, pthread_mutexattr_init — destroy and initialize the mutex
58537 attributes object

58538 **SYNOPSIS**

58539 ```
#include <pthread.h>
```

58540 ```
int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
```
58541 ```
int pthread_mutexattr_init(pthread_mutexattr_t *attr);
```

58542 **DESCRIPTION**

58543 The *pthread_mutexattr_destroy*( ) function shall destroy a mutex attributes object; the object
58544 becomes, in effect, uninitialized. An implementation may cause *pthread_mutexattr_destroy*( ) to
58545 set the object referenced by *attr* to an invalid value.

58546 A destroyed *attr* attributes object can be reinitialized using *pthread_mutexattr_init*( ); the results of
58547 otherwise referencing the object after it has been destroyed are undefined.

58548 The *pthread_mutexattr_init*( ) function shall initialize a mutex attributes object *attr* with the
58549 default value for all of the attributes defined by the implementation.

58550 Results are undefined if *pthread_mutexattr_init*( ) is called specifying an already initialized *attr*
58551 attributes object.

58552 After a mutex attributes object has been used to initialize one or more mutexes, any function
58553 affecting the attributes object (including destruction) shall not affect any previously initialized
58554 mutexes.

58555 The behavior is undefined if the value specified by the *attr* argument to
58556 *pthread_mutexattr_destroy*( ) does not refer to an initialized mutex attributes object.

58557 **RETURN VALUE**

58558 Upon successful completion, *pthread_mutexattr_destroy*( ) and *pthread_mutexattr_init*( ) shall
58559 return zero; otherwise, an error number shall be returned to indicate the error.

58560 **ERRORS**

58561 The *pthread_mutexattr_init*( ) function shall fail if:

58562 [ENOMEM] Insufficient memory exists to initialize the mutex attributes object.

58563 These functions shall not return an error code of [EINTR].

58564 **EXAMPLES**

58565 None.

58566 **APPLICATION USAGE**

58567 None.

58568 **RATIONALE**

58569 If an implementation detects that the value specified by the *attr* argument to
58570 *pthread_mutexattr_destroy*( ) does not refer to an initialized mutex attributes object, it is
58571 recommended that the function should fail and report an [EINVAL] error.

58572 See *pthread_attr_destroy( )* for a general explanation of attributes. Attributes objects allow
58573 implementations to experiment with useful extensions and permit extension of this volume of
58574 POSIX.1-2024 without changing the existing functions. Thus, they provide for future
58575 extensibility of this volume of POSIX.1-2024 and reduce the temptation to standardize
58576 prematurely on semantics that are not yet widely implemented or understood.

58577 Examples of possible additional mutex attributes that have been discussed are *spin_only*,
58578 *limited_spin*, *no_spin*, *recursive*, and *metered*. (To explain what the latter attributes might mean:

  
```

recursive mutexes would allow for multiple re-locking by the current owner; metered mutexes would transparently keep records of queue length, wait time, and so on.) Since there is not yet wide agreement on the usefulness of these resulting from shared implementation and usage experience, they are not yet specified in this volume of POSIX.1-2024. Mutex attributes objects, however, make it possible to test out these concepts for possible standardization at a later time.

Mutex Attributes and Performance

Care has been taken to ensure that the default values of the mutex attributes have been defined such that mutexes initialized with the defaults have simple enough semantics so that the locking and unlocking can be done with the equivalent of a test-and-set instruction (plus possibly a few other basic instructions).

There is at least one implementation method that can be used to reduce the cost of testing at lock-time if a mutex has non-default attributes. One such method that an implementation can employ (and this can be made fully transparent to fully conforming POSIX applications) is to secretly pre-lock any mutexes that are initialized to non-default attributes. Any later attempt to lock such a mutex causes the implementation to branch to the ``slow path'' as if the mutex were unavailable; then, on the slow path, the implementation can do the ``real work'' to lock a non-default mutex. The underlying unlock operation is more complicated since the implementation never really wants to release the pre-lock on this kind of mutex. This illustrates that, depending on the hardware, there may be certain optimizations that can be used so that whatever mutex attributes are considered ``most frequently used'' can be processed most efficiently.

Process Shared Memory and Synchronization

The existence of memory mapping functions in this volume of POSIX.1-2024 leads to the possibility that an application may allocate the synchronization objects from this section in memory that is accessed by multiple processes (and therefore, by threads of multiple processes).

In order to permit such usage, while at the same time keeping the usual case (that is, usage within a single process) efficient, a *process-shared* option has been defined.

If an implementation supports the _POSIX_THREAD_PROCESS_SHARED option, then the *process-shared* attribute can be used to indicate that mutexes or condition variables may be accessed by threads of multiple processes.

The default setting of PTHREAD_PROCESS_PRIVATE has been chosen for the *process-shared* attribute so that the most efficient forms of these synchronization objects are created by default.

Synchronization variables that are initialized with the PTHREAD_PROCESS_PRIVATE *process-shared* attribute may only be operated on by threads in the process that initialized them. Synchronization variables that are initialized with the PTHREAD_PROCESS_SHARED *process-shared* attribute may be operated on by any thread in any process that has access to it. In particular, these processes may exist beyond the lifetime of the initializing process. For example, the following code implements a simple counting semaphore in a mapped file that may be used by many processes.

```
/* sem.h */
struct semaphore {
    pthread_mutex_t lock;
    pthread_cond_t nonzero;
    unsigned count;
};
typedef struct semaphore semaphore_t;
```



```
58624       semaphore_t *semaphore_create(char *semaphore_name);
58625       semaphore_t *semaphore_open(char *semaphore_name);
58626       void semaphore_post(semaphore_t *semap);
58627       void semaphore_wait(semaphore_t *semap);
58628       void semaphore_close(semaphore_t *semap);

58629       /* sem.c */
58630       #include <sys/types.h>
58631       #include <sys/stat.h>
58632       #include <sys/mman.h>
58633       #include <fcntl.h>
58634       #include <pthread.h>
58635       #include "sem.h"

58636       semaphore_t *
58637       semaphore_create(char *semaphore_name)
58638       {
58639       int fd;
58640           semaphore_t *semap;
58641           pthread_mutexattr_t psharedm;
58642           pthread_condattr_t psharedc;

58643           fd = open(semaphore_name, O_RDWR | O_CREAT | O_EXCL, 0666);
58644           if (fd < 0)
58645               return (NULL);
58646           (void) ftruncate(fd, sizeof(semaphore_t));
58647           (void) pthread_mutexattr_init(&psharedm);
58648           (void) pthread_mutexattr_setpshared(&psharedm,
58649               PTHREAD_PROCESS_SHARED);
58650           (void) pthread_condattr_init(&psharedc);
58651           (void) pthread_condattr_setpshared(&psharedc,
58652               PTHREAD_PROCESS_SHARED);
58653           semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
58654                   PROT_READ | PROT_WRITE, MAP_SHARED,
58655                   fd, 0);
58656           close (fd);
58657           (void) pthread_mutex_init(&semap->lock, &psharedm);
58658           (void) pthread_cond_init(&semap->nonzero, &psharedc);
58659           semap->count = 0;
58660           return (semap);
58661       }

58662       semaphore_t *
58663       semaphore_open(char *semaphore_name)
58664       {
58665           int fd;
58666           semaphore_t *semap;

58667           fd = open(semaphore_name, O_RDWR, 0666);
58668           if (fd < 0)
58669               return (NULL);
58670           semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
58671                   PROT_READ | PROT_WRITE, MAP_SHARED,
58672                   fd, 0);
```



```
58673           close (fd);
58674           return (semap);
58675       }
58676       void
58677       semaphore_post(semaphore_t *semap)
58678       {
58679           pthread_mutex_lock(&semap->lock);
58680           if (semap->count == 0)
58681               pthread_cond_signal(&semapx->nonzero);
58682           semap->count++;
58683           pthread_mutex_unlock(&semap->lock);
58684       }
58685       void
58686       semaphore_wait(semaphore_t *semap)
58687       {
58688           pthread_mutex_lock(&semap->lock);
58689           while (semap->count == 0)
58690               pthread_cond_wait(&semap->nonzero, &semap->lock);
58691           semap->count--;
58692           pthread_mutex_unlock(&semap->lock);
58693       }
58694       void
58695       semaphore_close(semaphore_t *semap)
58696       {
58697           munmap((void *) semap, sizeof(semaphore_t));
58698       }
```

58699 The following code is for three separate processes that create, post, and wait on a semaphore in
58700 the file **/tmp/semaphore**. Once the file is created, the post and wait programs increment and
58701 decrement the counting semaphore (waiting and waking as required) even though they did not
58702 initialize the semaphore.

```
58703       /* create.c */
58704       #include "pthread.h"
58705       #include "sem.h"
58706       int
58707       main(void)
58708       {
58709           semaphore_t *semap;
58710           semap = semaphore_create("/tmp/semaphore");
58711           if (semap == NULL)
58712               exit(1);
58713           semaphore_close(semap);
58714           return (0);
58715       }
58716       /* post.c */
58717       #include "pthread.h"
58718       #include "sem.h"
58719       int
```



```
main(void)
{
    semaphore_t *semap;

    semap = semaphore_open("/tmp/semaphore");
    if (semap == NULL)
        exit(1);
    semaphore_post(semap);
    semaphore_close(semap);
    return (0);
}

/* wait.c */
#include "pthread.h"
#include "sem.h"

int
main(void)
{
    semaphore_t *semap;

    semap = semaphore_open("/tmp/semaphore");
    if (semap == NULL)
        exit(1);
    semaphore_wait(semap);
    semaphore_close(semap);
    return (0);
}
```

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_destroy( ), *pthread_create( )*, *pthread_mutex_destroy( )*

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_mutexattr_destroy( )* and *pthread_mutexattr_init( )* functions are marked as part of the Threads option.

IEEE PASC Interpretation 1003.1c #27 is applied, updating the ERRORS section.

Issue 7

The *pthread_mutexattr_destroy( )* and *pthread_mutexattr_init( )* functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

Issue 8

Austin Group Defect 1195 is applied, changing ``main()'' to ``main(void)''.

NAME

pthread_mutexattr_getprioceiling, pthread_mutexattr_setprioceiling — get and set the prioceiling attribute of the mutex attributes object (**REALTIME THREADS**)

SYNOPSIS

RPP|TPP

```
#include <pthread.h>

int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t
    *restrict attr, int *restrict prioceiling);
int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
    int prioceiling);
```

DESCRIPTION

The *pthread_mutexattr_getprioceiling*( ) and *pthread_mutexattr_setprioceiling*( ) functions, respectively, shall get and set the priority ceiling attribute of a mutex attributes object pointed to by *attr* which was previously created by the function *pthread_mutexattr_init*( ).

The *prioceiling* attribute contains the priority ceiling of initialized mutexes. The values of *prioceiling* are within the maximum range of priorities defined by SCHED_FIFO.

The *prioceiling* attribute defines the priority ceiling of initialized mutexes, which is the minimum priority level at which the critical section guarded by the mutex is executed. In order to avoid priority inversion, the priority ceiling of the mutex shall be set to a priority higher than or equal to the highest priority of all the threads that may lock that mutex. The values of *prioceiling* are within the maximum range of priorities defined under the SCHED_FIFO scheduling policy.

The behavior is undefined if the value specified by the *attr* argument to *pthread_mutexattr_getprioceiling*( ) or *pthread_mutexattr_setprioceiling*( ) does not refer to an initialized mutex attributes object.

RETURN VALUE

Upon successful completion, the *pthread_mutexattr_getprioceiling*( ) and *pthread_mutexattr_setprioceiling*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

[EINVAL] The value specified by *prioceiling* is invalid.

[EPERM] The caller does not have the privilege to perform the operation.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_mutexattr_getprioceiling*( ) or *pthread_mutexattr_setprioceiling*( ) does not refer to an initialized mutex attributes object, it is recommended that the function should fail and report an [EINVAL] error.

58803 **FUTURE DIRECTIONS**

58804 None.

58805 **SEE ALSO**

58806 *pthread_cond_destroy*( ), *pthread_create*( ), *pthread_mutex_destroy*( )

58807 XBD **<pthread.h>**

58808 **CHANGE HISTORY**

58809 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

58810 Marked as part of the Realtime Threads Feature Group.

58811 **Issue 6**

58812 The *pthread_mutexattr_getprioceiling*( ) and *pthread_mutexattr_setprioceiling*( ) functions are marked
58813 as part of the Threads and Thread Priority Protection options.

58814 The [ENOSYS] error condition has been removed as stubs need not be provided if an
58815 implementation does not support the Thread Priority Protection option.

58816 The [ENOTSUP] error condition has been removed since these functions do not have a *protocol*
58817 argument.

58818 The **restrict** keyword is added to the *pthread_mutexattr_getprioceiling*( ) prototype for alignment
58819 with the ISO/IEC 9899: 1999 standard.

58820 **Issue 7**

58821 The *pthread_mutexattr_getprioceiling*( ) and *pthread_mutexattr_setprioceiling*( ) functions are moved
58822 from the Threads option to require support of either the Robust Mutex Priority Protection option
58823 or the Non-Robust Mutex Priority Protection option.

58824 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition
58825 results in undefined behavior.

NAME

pthread_mutexattr_getprotocol, pthread_mutexattr_setprotocol — get and set the protocol attribute of the mutex attributes object (**REALTIME THREADS**)

SYNOPSIS

MC1
```
#include <pthread.h>

int pthread_mutexattr_getprotocol(const pthread_mutexattr_t
    *restrict attr, int *restrict protocol);
int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
    int protocol);
```

DESCRIPTION

The *pthread_mutexattr_getprotocol*( ) and *pthread_mutexattr_setprotocol*( ) functions, respectively, shall get and set the protocol attribute of a mutex attributes object pointed to by *attr* which was previously created by the function *pthread_mutexattr_init*( ).

The *protocol* attribute defines the protocol to be followed in utilizing mutexes. The value of *protocol* may be one of:

RPI|TPI PTHREAD_PRIO_INHERIT
MC1 PTHREAD_PRIO_NONE
RPP|TPP PTHREAD_PRIO_PROTECT

which are defined in the **<pthread.h>** header. The default value of the attribute shall be PTHREAD_PRIO_NONE.

When a thread owns a mutex with the PTHREAD_PRIO_NONE *protocol* attribute, its priority and scheduling shall not be affected by its mutex ownership.

RPI When a thread is blocking higher priority threads because of owning one or more robust mutexes with the PTHREAD_PRIO_INHERIT *protocol* attribute, it shall execute at the higher of its priority or the priority of the highest priority thread waiting on any of the robust mutexes owned by this thread and initialized with this protocol.

TPI When a thread is blocking higher priority threads because of owning one or more non-robust mutexes with the PTHREAD_PRIO_INHERIT *protocol* attribute, it shall execute at the higher of its priority or the priority of the highest priority thread waiting on any of the non-robust mutexes owned by this thread and initialized with this protocol.

RPP When a thread owns one or more robust mutexes initialized with the PTHREAD_PRIO_PROTECT protocol, it shall execute at the higher of its priority or the highest of the priority ceilings of all the robust mutexes owned by this thread and initialized with this attribute, regardless of whether other threads are blocked on any of these robust mutexes or not.

TPP When a thread owns one or more non-robust mutexes initialized with the PTHREAD_PRIO_PROTECT protocol, it shall execute at the higher of its priority or the highest of the priority ceilings of all the non-robust mutexes owned by this thread and initialized with this attribute, regardless of whether other threads are blocked on any of these non-robust mutexes or not.

If a thread simultaneously owns several mutexes initialized with different protocols, it shall execute at the highest of the priorities that it would have obtained by each of these protocols.

RPI | TPI When a thread makes a call to *pthread_mutex_lock*( ), the mutex was initialized with the protocol attribute having the value PTHREAD_PRIO_INHERIT, when the calling thread is blocked because the mutex is owned by another thread, that owner thread shall inherit the priority level of the calling thread as long as it continues to own the mutex. The implementation shall update its execution priority to the maximum of its assigned priority and all its inherited priorities. Furthermore, if this owner thread itself becomes blocked on another mutex with the *protocol* attribute having the value PTHREAD_PRIO_INHERIT, the same priority inheritance effect shall be propagated to this other owner thread, in a recursive manner.

The behavior is undefined if the value specified by the *attr* argument to *pthread_mutexattr_getprotocol*( ) or *pthread_mutexattr_setprotocol*( ) does not refer to an initialized mutex attributes object.

RETURN VALUE

Upon successful completion, the *pthread_mutexattr_getprotocol*( ) and *pthread_mutexattr_setprotocol*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutexattr_setprotocol*( ) function shall fail if:

[ENOTSUP] The value specified by *protocol* is an unsupported value.

The *pthread_mutexattr_getprotocol*( ) and *pthread_mutexattr_setprotocol*( ) functions may fail if:

[EINVAL] The value specified by *protocol* is invalid.

[EPERM] The caller does not have the privilege to perform the operation.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_mutexattr_getprotocol*( ) or *pthread_mutexattr_setprotocol*( ) does not refer to an initialized mutex attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_destroy( ), *pthread_create*( ), *pthread_mutex_destroy*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

Issue 6

The *pthread_mutexattr_getprotocol*( ) and *pthread_mutexattr_setprotocol*( ) functions are marked as part of the Threads option and either the Thread Priority Protection or Thread Priority Inheritance options.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Priority Protection or Thread Priority Inheritance options.

The **restrict** keyword is added to the *pthread_mutexattr_getprotocol*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

SD5-XSH-ERN-135 is applied, updating the DESCRIPTION to define a default value for the *protocol* attribute.

SD5-XSH-ERN-188 is applied, updating the DESCRIPTION.

The *pthread_mutexattr_getprotocol*( ) and *pthread_mutexattr_setprotocol*( ) functions are moved from the Threads option to require support of either the Non-Robust Mutex Priority Protection option or the Non-Robust Mutex Priority Inheritance option or the Robust Mutex Priority Protection option or the Robust Mutex Priority Inheritance option.

The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

Issue 8

Austin Group Defect 1610 is applied, moving text relating to the effects of PTHREAD_PRIO_INHERIT and PTHREAD_PRIO_PROTECT on scheduling queues to Scheduling Policies (on page 531).

NAME

pthread_mutexattr_getpshared, pthread_mutexattr_setpshared — get and set the process-shared attribute

SYNOPSIS

TSH

```
#include <pthread.h>

int pthread_mutexattr_getpshared(const pthread_mutexattr_t
    *restrict attr, int *restrict pshared);
int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
    int pshared);
```

DESCRIPTION

The *pthread_mutexattr_getpshared*( ) function shall obtain the value of the *process-shared* attribute from the attributes object referenced by *attr*.

The *pthread_mutexattr_setpshared*( ) function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

The *process-shared* attribute is set to PTHREAD_PROCESS_SHARED to permit a mutex to be operated upon by any thread that has access to the memory where the mutex is allocated, even if the mutex is allocated in memory that is shared by multiple processes. See Section 2.9.9 (on page 548) for further requirements. The default value of the attribute shall be PTHREAD_PROCESS_PRIVATE.

The behavior is undefined if the value specified by the *attr* argument to *pthread_mutexattr_getpshared*( ) or *pthread_mutexattr_setpshared*( ) does not refer to an initialized mutex attributes object.

RETURN VALUE

Upon successful completion, *pthread_mutexattr_setpshared*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

Upon successful completion, *pthread_mutexattr_getpshared*( ) shall return zero and store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutexattr_setpshared*( ) function may fail if:

[EINVAL] The new value specified for the attribute is outside the range of legal values for that attribute.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_mutexattr_getpshared*( ) or *pthread_mutexattr_setpshared*( ) does not refer to an initialized mutex attributes object, it is recommended that the function should fail and report an [EINVAL] error.

58971 **FUTURE DIRECTIONS**

58972 None.

58973 **SEE ALSO**

58974 *pthread_cond_destroy*( ), *pthread_create*( ), *pthread_mutex_destroy*( ), *pthread_mutexattr_destroy*( )

58975 XBD **<pthread.h>**

58976 **CHANGE HISTORY**

58977 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

58978 **Issue 6**

58979 The *pthread_mutexattr_getpshared*( ) and *pthread_mutexattr_setpshared*( ) functions are marked as
58980 part of the Threads and Thread Process-Shared Synchronization options.

58981 The **restrict** keyword is added to the *pthread_mutexattr_getpshared*( ) prototype for alignment
58982 with the ISO/IEC 9899: 1999 standard.

58983 **Issue 7**

58984 The *pthread_mutexattr_getpshared*( ) and *pthread_mutexattr_setpshared*( ) functions are marked only
58985 as part of the Thread Process-Shared Synchronization option as the Threads option is now part
58986 of the Base.

58987 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition
58988 results in undefined behavior.

58989 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0281 [972] and XSH/TC2-2008/0282
58990 [757] are applied.

NAME

pthread_mutexattr_getrobust, pthread_mutexattr_setrobust — get and set the mutex robust attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_mutexattr_getrobust(const pthread_mutexattr_t *restrict
    attr, int *restrict robust);
int pthread_mutexattr_setrobust(pthread_mutexattr_t *attr,
    int robust);
```

DESCRIPTION

The *pthread_mutexattr_getrobust*( ) and *pthread_mutexattr_setrobust*( ) functions, respectively, shall get and set the mutex *robust* attribute. This attribute is set in the *robust* parameter. Valid values for *robust* include:

PTHREAD_MUTEX_STALLED
: No special actions are taken if the owner of the mutex is terminated while holding the mutex lock. This can lead to deadlocks if no other thread can unlock the mutex.
This is the default value.

PTHREAD_MUTEX_ROBUST
: If the process containing the owning thread of a robust mutex terminates while holding the mutex lock, the next thread that acquires the mutex shall be notified about the termination by the return value [EOWNERDEAD] from the locking function. If the owning thread of a robust mutex terminates while holding the mutex lock, the next thread that attempts to acquire the mutex may be notified about the termination by the return value [EOWNERDEAD]. The notified thread can then attempt to make the state protected by the mutex consistent again, and if successful can mark the mutex state as consistent by calling *pthread_mutex_consistent*( ). After a subsequent successful call to *pthread_mutex_unlock*( ), the mutex lock shall be released and can be used normally by other threads. If the mutex is unlocked without a call to *pthread_mutex_consistent*( ), it shall be in a permanently unusable state and all attempts to lock the mutex shall fail with the error [ENOTRECOVERABLE]. The only permissible operation on such a mutex is *pthread_mutex_destroy*( ).

The behavior is undefined if the value specified by the *attr* argument to *pthread_mutexattr_getrobust*( ) or *pthread_mutexattr_setrobust*( ) does not refer to an initialized mutex attributes object.

RETURN VALUE

Upon successful completion, the *pthread_mutexattr_getrobust*( ) function shall return zero and store the value of the *robust* attribute of *attr* into the object referenced by the *robust* parameter. Otherwise, an error value shall be returned to indicate the error. If successful, the *pthread_mutexattr_setrobust*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutexattr_setrobust*( ) function shall fail if:

[EINVAL] The value of *robust* is invalid.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

The actions required to make the state protected by the mutex consistent again are solely dependent on the application. If it is not possible to make the state of a mutex consistent, robust mutexes can be used to notify this situation by calling *pthread_mutex_unlock*( ) without a prior call to *pthread_mutex_consistent*( ).

If the state is declared inconsistent by calling *pthread_mutex_unlock*( ) without a prior call to *pthread_mutex_consistent*( ), a possible approach could be to destroy the mutex and then reinitialize it. However, it should be noted that this is possible only in certain situations where the state protected by the mutex has to be reinitialized and coordination achieved with other threads blocked on the mutex, because otherwise a call to a locking function with a reference to a mutex object invalidated by a call to *pthread_mutex_destroy*( ) results in undefined behavior.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_mutexattr_getrobust*( ) or *pthread_mutexattr_setrobust*( ) does not refer to an initialized mutex attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_mutex_consistent( ), *pthread_mutex_destroy*( ), *pthread_mutex_lock*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 7.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0283 [748] is applied.

NAME

pthread_mutexattr_gettype, pthread_mutexattr_settype — get and set the mutex type attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict attr,
    int *restrict type);
int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);
```

DESCRIPTION

The *pthread_mutexattr_gettype*( ) and *pthread_mutexattr_settype*( ) functions, respectively, shall get and set the mutex *type* attribute. This attribute is set in the *type* parameter to these functions. The default value of the *type* attribute is PTHREAD_MUTEX_DEFAULT.

The type of mutex is contained in the *type* attribute of the mutex attributes. Valid mutex types include:

PTHREAD_MUTEX_NORMAL
PTHREAD_MUTEX_ERRORCHECK
PTHREAD_MUTEX_RECURSIVE
PTHREAD_MUTEX_DEFAULT

The mutex type affects the behavior of calls which lock and unlock the mutex. See *pthread_mutex_lock*( ) for details. An implementation may map PTHREAD_MUTEX_DEFAULT to one of the other mutex types.

The behavior is undefined if the value specified by the *attr* argument to *pthread_mutexattr_gettype*( ) or *pthread_mutexattr_settype*( ) does not refer to an initialized mutex attributes object.

RETURN VALUE

Upon successful completion, the *pthread_mutexattr_gettype*( ) function shall return zero and store the value of the *type* attribute of *attr* into the object referenced by the *type* parameter. Otherwise, an error shall be returned to indicate the error.

If successful, the *pthread_mutexattr_settype*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutexattr_settype*( ) function shall fail if:

[EINVAL] The value *type* is invalid.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

It is advised that an application should not use a PTHREAD_MUTEX_RECURSIVE mutex with condition variables because the implicit unlock performed in a *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), or *pthread_cond_wait*( ) call may not actually release the mutex (if it had been locked multiple times). If this happens, no other thread can satisfy the condition of the predicate.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_mutexattr_gettype*( ) or *pthread_mutexattr_settype*( ) does not refer to an initialized mutex attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_clockwait( ), *pthread_mutex_lock*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5.

Issue 6

The Open Group Corrigendum U033/3 is applied. The SYNOPSIS for *pthread_mutexattr_gettype*( ) is updated so that the first argument is of type **const pthread_mutexattr_t** *.

The **restrict** keyword is added to the *pthread_mutexattr_gettype*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

The *pthread_mutexattr_gettype*( ) and *pthread_mutexattr_settype*( ) functions are moved from the XSI option to the Base.

The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0464 [121] is applied.

Issue 8

Austin Group Defect 1216 is applied, adding *pthread_cond_clockwait*( ).

59126 **NAME**

59127 pthread_mutexattr_init — initialize the mutex attributes object

59128 **SYNOPSIS**

59129 ```
#include <pthread.h>
```

59130 ```
int pthread_mutexattr_init(pthread_mutexattr_t *attr);
```

59131 **DESCRIPTION**

59132 Refer to *pthread_mutexattr_destroy( )*.

NAME

pthread_mutexattr_setprioceiling — set the prioceiling attribute of the mutex attributes object (**REALTIME THREADS**)

SYNOPSIS

RPP|TPP

```
#include <pthread.h>

int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
    int prioceiling);
```

DESCRIPTION

Refer to *pthread_mutexattr_getprioceiling( )*.

NAME

pthread_mutexattr_setprotocol — set the protocol attribute of the mutex attributes object (**REALTIME THREADS**)

SYNOPSIS

MC1

```
#include <pthread.h>

int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
    int protocol);
```

DESCRIPTION

Refer to *pthread_mutexattr_getprotocol( )*.

NAME

pthread_mutexattr_setpshared — set the process-shared attribute

SYNOPSIS

TSH
```
#include <pthread.h>

int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
    int pshared);
```

DESCRIPTION

Refer to *pthread_mutexattr_getpshared*( ).

59159 **NAME**

59160 pthread_mutexattr_setrobust — get and set the mutex robust attribute

59161 **SYNOPSIS**

59162 ```
#include <pthread.h>
```

59163 ```
int pthread_mutexattr_setrobust(pthread_mutexattr_t *attr,
```

59164 ```
    int robust);
```

59165 **DESCRIPTION**

59166 Refer to *pthread_mutexattr_getrobust( )*.

  
```

59167 **NAME**

59168 pthread_mutexattr_settype — set the mutex type attribute

59169 **SYNOPSIS**

59170 `#include <pthread.h>`

59171 `int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);`

59172 **DESCRIPTION**

59173 Refer to *pthread_mutexattr_gettype( )*.

NAME

pthread_once — dynamic package initialization

SYNOPSIS

```
#include <pthread.h>

int pthread_once(pthread_once_t *once_control,
    void (*init_routine)(void));
pthread_once_t once_control = PTHREAD_ONCE_INIT;
```

DESCRIPTION

The first call to *pthread_once*( ) by any thread in a process, with a given *once_control*, shall call the *init_routine* with no arguments. Subsequent calls of *pthread_once*( ) with the same *once_control* shall not call the *init_routine*. On return from *pthread_once*( ), *init_routine* shall have completed. The *once_control* parameter shall determine whether the associated initialization routine has been called.

The *pthread_once*( ) function is not a cancellation point. However, if *init_routine* is a cancellation point and is canceled, the effect on *once_control* shall be as if *pthread_once*( ) was never called.

If the call to *init_routine* is terminated by a call to *longjmp*( ) or *siglongjmp*( ), the behavior is undefined.

The constant PTHREAD_ONCE_INIT is defined in the **<pthread.h>** header.

The behavior of *pthread_once*( ) is undefined if *once_control* has automatic storage duration or is not initialized by PTHREAD_ONCE_INIT.

RETURN VALUE

Upon successful completion, *pthread_once*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_once*( ) function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

If *init_routine* recursively calls *pthread_once*( ) with the same *once_control*, the recursive call will not call the specified *init_routine*, and thus the specified *init_routine* will not complete, and thus the recursive call to *pthread_once*( ) will not return. Use of *longjmp*( ) or *siglongjmp*( ) within an *init_routine* to jump to a point outside of *init_routine* prevents *init_routine* from returning.

RATIONALE

Some C libraries are designed for dynamic initialization. That is, the global initialization for the library is performed when the first procedure in the library is called. In a single-threaded program, this is normally implemented using a static variable whose value is checked on entry to a routine, as follows:

```
static int random_is_initialized = 0;
extern void initialize_random(void);

int random_function()
{
    if (random_is_initialized == 0) {
        initialize_random();
        random_is_initialized = 1;
    }
```



```
    ... /* Operations performed after initialization. */
}
```

To keep the same structure in a multi-threaded program, a new primitive is needed. Otherwise, library initialization has to be accomplished by an explicit call to a library-exported initialization function prior to any use of the library.

For dynamic library initialization in a multi-threaded process, if an initialization flag is used the flag needs to be protected against modification by multiple threads simultaneously calling into the library. This can be done by using a mutex (initialized by assigning PTHREAD_MUTEX_INITIALIZER). However, the better solution is to use *pthread_once*( ) which is designed for exactly this purpose, as follows:

```
#include <pthread.h>
static pthread_once_t random_is_initialized = PTHREAD_ONCE_INIT;
extern void initialize_random(void);

int random_function()
{
    (void) pthread_once(&random_is_initialized, initialize_random);
    ... /* Operations performed after initialization. */
}
```

If an implementation detects that the value specified by the *once_control* argument to *pthread_once*( ) does not refer to a **pthread_once_t** object initialized by PTHREAD_ONCE_INIT, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_once*( ) function is marked as part of the Threads option.

The [EINVAL] error is added as a ``may fail'' case for if either argument is invalid.

Issue 7

The *pthread_once*( ) function is moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized **pthread_once_t** object is removed; this condition results in undefined behavior.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0284 [863], XSH/TC2-2008/0285 [874], XSH/TC2-2008/0286 [874], and XSH/TC2-2008/0287 [747] are applied.

Issue 8

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

NAME

pthread_rwlock_clockrdlock, pthread_rwlock_timedrdlock — lock a read-write lock for reading

SYNOPSIS

```
#include <pthread.h>

int pthread_rwlock_clockrdlock(pthread_rwlock_t *restrict rwlock,
    clockid_t clock_id, const struct timespec *restrict abstime);
int pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict rwlock,
    const struct timespec *restrict abstime);
```

DESCRIPTION

The *pthread_rwlock_clockrdlock*( ) and *pthread_rwlock_timedrdlock*( ) functions shall apply a read lock to the read-write lock referenced by *rwlock* as in the *pthread_rwlock_rdlock*( ) function. However, if the lock cannot be acquired without waiting for other threads to unlock the lock, this wait shall be terminated when the specified timeout expires. The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the call.

For *pthread_rwlock_timedrdlock*( ), the timeout shall be based on the CLOCK_REALTIME clock. For *pthread_rwlock_clockrdlock*( ), the timeout shall be based on the clock specified by the *clock_id* argument. The resolution of the timeout shall be the resolution of the clock on which it is based. Implementations shall support passing CLOCK_REALTIME and CLOCK_MONOTONIC to *pthread_rwlock_clockrdlock*( ) as the *clock_id* argument.

Under no circumstances shall the function fail with a timeout if the lock can be acquired immediately. The validity of the *abstime* parameter need not be checked if the lock can be immediately acquired.

If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-write lock via a call to *pthread_rwlock_clockrdlock*( ) or *pthread_rwlock_timedrdlock*( ), upon return from the signal handler the thread shall resume waiting for the lock as if it was not interrupted.

The calling thread may deadlock if at the time the call is made it holds a write lock on *rwlock*. The results are undefined if these functions are called with an uninitialized read-write lock.

RETURN VALUE

The *pthread_rwlock_clockrdlock*( ) and *pthread_rwlock_timedrdlock*( ) functions shall return zero if the lock for reading on the read-write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlock_clockrdlock*( ) and *pthread_rwlock_timedrdlock*( ) functions shall fail if:

[ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

The *pthread_rwlock_clockrdlock*( ) and *pthread_rwlock_timedrdlock*( ) functions may fail if:

[EAGAIN] The read lock could not be acquired because the maximum number of read locks for lock would be exceeded.

[EDEADLK] A deadlock condition was detected or the calling thread already holds a write lock on *rwlock*.

[EINVAL] The *abstime* nanosecond value is less than zero or greater than or equal to 1 000 million, or the *pthread_rwlock_clockrdlock*( ) function was passed an invalid or unsupported *clock_id* value.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in XBD Section 3.275 (on page 72).

RATIONALE

If an implementation detects that the value specified by the *rwlock* argument to *pthread_rwlock_clockrdlock*( ) or *pthread_rwlock_timedrdlock*( ) does not refer to an initialized read-write lock object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_clockwrlock( ), *pthread_rwlock_destroy*( ), *pthread_rwlock_rdlock*( ), *pthread_rwlock_trywrlock*( ), *pthread_rwlock_unlock*( )

XBD Section 3.275 (on page 72), Section 4.15.2 (on page 104), **<pthread.h>**, **<time.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/102 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

Issue 7

The *pthread_rwlock_timedrdlock*( ) function is moved from the Timeouts option to the Base.

The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results in undefined behavior.

Issue 8

Austin Group Defect 592 is applied, removing text relating to **<time.h>** from the SYNOPSIS and DESCRIPTION sections.

Austin Group Defects 1216 and 1472 are applied, adding *pthread_rwlock_clockrdlock*( ).

NAME

pthread_rwlock_clockwrlock, pthread_rwlock_timedwrlock — lock a read-write lock for writing

SYNOPSIS

```
#include <pthread.h>

int pthread_rwlock_clockwrlock(pthread_rwlock_t *restrict rwlock,
    clockid_t clock_id, const struct timespec *restrict abstime);
int pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict rwlock,
    const struct timespec *restrict abstime);
```

DESCRIPTION

The *pthread_rwlock_clockwrlock*( ) and *pthread_rwlock_timedwrlock*( ) functions shall apply a write lock to the read-write lock referenced by *rwlock* as in the *pthread_rwlock_wrlock*( ) function. However, if the lock cannot be acquired without waiting for other threads to unlock the lock, this wait shall be terminated when the specified timeout expires. The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the call.

For *pthread_rwlock_timedwrlock*( ), the timeout shall be based on the CLOCK_REALTIME clock. For *pthread_rwlock_clockwrlock*( ), the timeout shall be based on the clock specified by the *clock_id* argument. The resolution of the timeout shall be the resolution of the clock on which it is based. Implementations shall support passing CLOCK_REALTIME and CLOCK_MONOTONIC to *pthread_rwlock_clockwrlock*( ) as the *clock_id* argument.

Under no circumstances shall the function fail with a timeout if the lock can be acquired immediately. The validity of the *abstime* parameter need not be checked if the lock can be immediately acquired.

If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-write lock via a call to *pthread_rwlock_clockwrlock*( ) or *pthread_rwlock_timedwrlock*( ), upon return from the signal handler the thread shall resume waiting for the lock as if it was not interrupted.

The calling thread may deadlock if at the time the call is made it holds the read-write lock. The results are undefined if these functions are called with an uninitialized read-write lock.

RETURN VALUE

The *pthread_rwlock_clockwrlock*( ) and *pthread_rwlock_timedwrlock*( ) functions shall return zero if the lock for writing on the read-write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlock_clockwrlock*( ) and *pthread_rwlock_timedwrlock*( ) functions shall fail if:

[ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

The *pthread_rwlock_clockwrlock*( ) and *pthread_rwlock_timedwrlock*( ) functions may fail if:

[EDEADLK] A deadlock condition was detected or the calling thread already holds the *rwlock*.

[EINVAL] The *abstime* nanosecond value is less than zero or greater than or equal to 1 000 million, or the *pthread_rwlock_clockwrlock*( ) function was passed an invalid or unsupported *clock_id* value.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in XBD Section 3.275 (on page 72).

RATIONALE

If an implementation detects that the value specified by the *rwlock* argument to *pthread_rwlock_clockwrlock*( ) or *pthread_rwlock_timedwrlock*( ) does not refer to an initialized read-write lock object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_clockrdlock( ), *pthread_rwlock_destroy*( ), *pthread_rwlock_rdlock*( ), *pthread_rwlock_trywrlock*( ), *pthread_rwlock_unlock*( )

XBD Section 3.275 (on page 72), Section 4.15.2 (on page 104), **<pthread.h>**, **<time.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/103 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

Issue 7

The *pthread_rwlock_timedwrlock*( ) function is moved from the Timeouts option to the Base.

The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results in undefined behavior.

Issue 8

Austin Group Defect 592 is applied, removing text relating to **<time.h>** from the SYNOPSIS and DESCRIPTION sections.

Austin Group Defects 1216 and 1472 are applied, adding *pthread_rwlock_clockwrlock*( ).

NAME

pthread_rwlock_destroy, pthread_rwlock_init — destroy and initialize a read-write lock object

SYNOPSIS

```
#include <pthread.h>

int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
int pthread_rwlock_init(pthread_rwlock_t *restrict rwlock,
    const pthread_rwlockattr_t *restrict attr);
pthread_rwlock_t rwlock = PTHREAD_RWLOCK_INITIALIZER;
```

DESCRIPTION

The *pthread_rwlock_destroy*( ) function shall destroy the read-write lock object referenced by *rwlock* and release any resources used by the lock. The effect of subsequent use of the lock is undefined until the lock is reinitialized by another call to *pthread_rwlock_init*( ). An implementation may cause *pthread_rwlock_destroy*( ) to set the object referenced by *rwlock* to an invalid value. Results are undefined if *pthread_rwlock_destroy*( ) is called when any thread holds *rwlock*. Attempting to destroy an uninitialized read-write lock results in undefined behavior.

The *pthread_rwlock_init*( ) function shall allocate any resources required to use the read-write lock referenced by *rwlock* and initializes the lock to an unlocked state with attributes referenced by *attr*. If *attr* is NULL, the default read-write lock attributes shall be used; the effect is the same as passing the address of a default read-write lock attributes object. Once initialized, the lock can be used any number of times without being reinitialized. Results are undefined if *pthread_rwlock_init*( ) is called specifying an already initialized read-write lock. Results are undefined if a read-write lock is used without first being initialized.

If the *pthread_rwlock_init*( ) function fails, *rwlock* shall not be initialized and the contents of *rwlock* are undefined.

See Section 2.9.9 (on page 548) for further requirements.

In cases where default read-write lock attributes are appropriate, the macro PTHREAD_RWLOCK_INITIALIZER can be used to initialize read-write locks. The effect shall be equivalent to dynamic initialization by a call to *pthread_rwlock_init*( ) with the *attr* parameter specified as NULL, except that no error checks are performed.

The behavior is undefined if the value specified by the *attr* argument to *pthread_rwlock_init*( ) does not refer to an initialized read-write lock attributes object.

RETURN VALUE

If successful, the *pthread_rwlock_destroy*( ) and *pthread_rwlock_init*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlock_init*( ) function shall fail if:

[EAGAIN] The system lacked the necessary resources (other than memory) to initialize another read-write lock.

[ENOMEM] Insufficient memory exists to initialize the read-write lock.

[EPERM] The caller does not have the privilege to perform the operation.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications using these and related read-write lock functions may be subject to priority inversion, as discussed in XBD Section 3.275 (on page 72).

RATIONALE

If an implementation detects that the value specified by the *rwlock* argument to *pthread_rwlock_destroy*( ) does not refer to an initialized read-write lock object, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the *attr* argument to *pthread_rwlock_init*( ) does not refer to an initialized read-write lock attributes object, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the *rwlock* argument to *pthread_rwlock_destroy*( ) or *pthread_rwlock_init*( ) refers to a locked read-write lock object, or detects that the value specified by the *rwlock* argument to *pthread_rwlock_init*( ) refers to an already initialized read-write lock object, it is recommended that the function should fail and report an [EBUSY] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_clockrdlock( ), *pthread_rwlock_clockwrlock*( ), *pthread_rwlock_rdlock*( ), *pthread_rwlock_trywrlock*( ), *pthread_rwlock_unlock*( )

XBD Section 3.275 (on page 72), **<pthread.h>**

CHANGE HISTORY

First released in Issue 5.

Issue 6

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension). The initializer macro is also deleted from the SYNOPSIS.
- The DESCRIPTION is updated as follows:
  - — It explicitly notes allocation of resources upon initialization of a read-write lock object.
  - — A paragraph is added specifying that copies of read-write lock objects may not be used.
- An [EINVAL] error is added to the ERRORS section for *pthread_rwlock_init*( ), indicating that the *rwlock* value is invalid.
- The SEE ALSO section is updated.

The **restrict** keyword is added to the *pthread_rwlock_init*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/45 is applied, adding APPLICATION USAGE relating to priority inversion.

Issue 7

Austin Group Interpretation 1003.1-2001 #048 is applied, adding the PTHREAD_RWLOCK_INITIALIZER macro.

The *pthread_rwlock_destroy*( ) and *pthread_rwlock_init*( ) functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized read-write lock object or read-write lock attributes object is removed; this condition results in undefined behavior.

The [EBUSY] error for a locked read-write lock object or an already initialized read-write lock object is removed; this condition results in undefined behavior.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0465 [70] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0288 [972] and XSH/TC2-2008/0289 [758] are applied.

59495 **NAME**

59496 pthread_rwlock_rdlock, pthread_rwlock_tryrdlock — lock a read-write lock object for reading

59497 **SYNOPSIS**

59498
```
#include <pthread.h>
```

59499
```
int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);
```
59500
```
int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);
```

59501 **DESCRIPTION**

59502 The *pthread_rwlock_rdlock*( ) function shall apply a read lock to the read-write lock referenced by
59503 *rwlock*. The calling thread acquires the read lock if a writer does not hold the lock and there are
59504 no writers blocked on the lock.

59505 TPS If the Thread Execution Scheduling option is supported, and the threads that hold or are blocked
59506 on the lock are executing with the scheduling policies SCHED_FIFO or SCHED_RR, the calling
59507 thread shall not acquire the lock if a writer holds the lock or if the calling thread does not
59508 already hold a read lock and writers of higher or equal priority are blocked on the lock;
59509 otherwise, the calling thread shall acquire the lock.

59510 TPS TSP If the Thread Execution Scheduling option is supported, and the threads that hold or are blocked
59511 on the lock are executing with the SCHED_SPORADIC scheduling policy, the calling thread
59512 shall not acquire the lock if a writer holds the lock or if the calling thread does not already hold a
59513 read lock and writers of higher or equal priority are blocked on the lock; otherwise, the calling
59514 thread shall acquire the lock.

59515 If the Thread Execution Scheduling option is not supported, it is implementation-defined
59516 whether the calling thread acquires the lock when a writer does not hold the lock and there are
59517 writers blocked on the lock. If a writer holds the lock, the calling thread shall not acquire the
59518 read lock.

59519 Whether the Thread Execution Scheduling option is supported or not, if the read lock is not
59520 acquired, the calling thread shall block until it can acquire the lock. The calling thread may
59521 deadlock if at the time the call is made it holds a write lock.

59522 A thread may hold multiple concurrent read locks on *rwlock* (that is, successfully call the
59523 *pthread_rwlock_rdlock*( ) function *n* times). If so, the application shall ensure that the thread
59524 performs matching unlocks (that is, it calls the *pthread_rwlock_unlock*( ) function *n* times).

59525 The maximum number of simultaneous read locks that an implementation guarantees can be
59526 applied to a read-write lock shall be implementation-defined. The *pthread_rwlock_rdlock*( )
59527 function may fail if this maximum would be exceeded.

59528 The *pthread_rwlock_tryrdlock*( ) function shall apply a read lock as in the *pthread_rwlock_rdlock*( )
59529 function, with the exception that the function shall fail if the equivalent *pthread_rwlock_rdlock*( )
59530 call would have blocked the calling thread. In no case shall the *pthread_rwlock_tryrdlock*( )
59531 function ever block; it always either acquires the lock or fails and returns immediately.

59532 Results are undefined if any of these functions are called with an uninitialized read-write lock.

59533 If a signal is delivered to a thread waiting for a read-write lock for reading, upon return from the
59534 signal handler the thread resumes waiting for the read-write lock for reading as if it was not
59535 interrupted.

59536 **RETURN VALUE**

59537 If successful, the *pthread_rwlock_rdlock*( ) function shall return zero; otherwise, an error number
59538 shall be returned to indicate the error.

59539 The *pthread_rwlock_tryrdlock*( ) function shall return zero if the lock for reading on the read-write

lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlock_tryrdlock*( ) function shall fail if:

[EBUSY] The read-write lock could not be acquired for reading because a writer holds the lock or a writer with the appropriate priority was blocked on it.

The *pthread_rwlock_rdlock*( ) and *pthread_rwlock_tryrdlock*( ) functions may fail if:

[EAGAIN] The read lock could not be acquired because the maximum number of read locks for *rwlock* has been exceeded.

The *pthread_rwlock_rdlock*( ) function may fail if:

[EDEADLK] A deadlock condition was detected or the current thread already owns the read-write lock for writing.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in XBD Section 3.275 (on page 72).

RATIONALE

If an implementation detects that the value specified by the *rwlock* argument to *pthread_rwlock_rdlock*( ) or *pthread_rwlock_tryrdlock*( ) does not refer to an initialized read-write lock object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_clockrdlock( ), *pthread_rwlock_clockwrlock*( ), *pthread_rwlock_destroy*( ), *pthread_rwlock_trywrlock*( ), *pthread_rwlock_unlock*( )

XBD Section 3.275 (on page 72), Section 4.15.2 (on page 104), **<pthread.h>**

CHANGE HISTORY

First released in Issue 5.

Issue 6

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- The DESCRIPTION is updated as follows:
  - Conditions under which writers have precedence over readers are specified.
  - Failure of *pthread_rwlock_tryrdlock*( ) is clarified.
  - A paragraph on the maximum number of read locks is added.

- In the ERRORS sections, [EBUSY] is modified to take into account write priority, and [EDEADLK] is deleted as a *pthread_rwlock_tryrdlock*( ) error.
- The SEE ALSO section is updated.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/101 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

Issue 7

The *pthread_rwlock_rdlock*( ) and *pthread_rwlock_tryrdlock*( ) functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results in undefined behavior.

Issue 8

Austin Group Defect 1111 is applied, changing ``threads involved in the lock'' to ``threads that hold or are blocked on the lock'' and clarifying that a requirement when the read lock is not acquired applies regardless of whether or not the Thread Execution Scheduling option is supported.

NAME

pthread_rwlock_timedrdlock — lock a read-write lock for reading

SYNOPSIS

```
#include <pthread.h>

int pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict rwlock,
    const struct timespec *restrict abstime);
```

DESCRIPTION

Refer to *pthread_rwlock_clockrdlock( )*.

59602 **NAME**

59603 pthread_rwlock_timedwrlock — lock a read-write lock for writing

59604 **SYNOPSIS**

59605 ```
#include <pthread.h>
```

59606 ```
int pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict rwlock,
```

59607 ```
    const struct timespec *restrict abstime);
```

59608 **DESCRIPTION**

59609 Refer to *pthread_rwlock_clockwrlock( )*.

  
```

NAME

pthread_rwlock_tryrdlock — lock a read-write lock object for reading

SYNOPSIS

```
#include <pthread.h>

int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);
```

DESCRIPTION

Refer to *pthread_rwlock_rdlock( )*.

NAME

pthread_rwlock_trywrlock, pthread_rwlock_wrlock — lock a read-write lock object for writing

SYNOPSIS

```
#include <pthread.h>

int pthread_rwlock_trywrlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);
```

DESCRIPTION

The *pthread_rwlock_trywrlock*( ) function shall apply a write lock like the *pthread_rwlock_wrlock*( ) function, with the exception that the function shall fail if any thread currently holds *rwlock* (for reading or writing).

The *pthread_rwlock_wrlock*( ) function shall apply a write lock to the read-write lock referenced by *rwlock*. The calling thread shall acquire the write lock if no thread (reader or writer) holds the read-write lock *rwlock*. Otherwise, if another thread holds the read-write lock *rwlock*, the calling thread shall block until it can acquire the lock. If a deadlock condition occurs or the calling thread already owns the read-write lock for writing or reading, the call shall either deadlock or return [EDEADLK].

Results are undefined if any of these functions are called with an uninitialized read-write lock.

If a signal is delivered to a thread waiting for a read-write lock for writing, upon return from the signal handler the thread resumes waiting for the read-write lock for writing as if it was not interrupted.

RETURN VALUE

The *pthread_rwlock_trywrlock*( ) function shall return zero if the lock for writing on the read-write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to indicate the error.

If successful, the *pthread_rwlock_wrlock*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlock_trywrlock*( ) function shall fail if:

[EBUSY] The read-write lock could not be acquired for writing because it was already locked for reading or writing.

The *pthread_rwlock_wrlock*( ) function may fail if:

[EDEADLK] A deadlock condition was detected or the current thread already owns the read-write lock for writing or reading.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in XBD Section 3.275 (on page 72).

RATIONALE

If an implementation detects that the value specified by the *rwlock* argument to *pthread_rwlock_trywrlock*( ) or *pthread_rwlock_wrlock*( ) does not refer to an initialized read-write lock object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_clockrdlock( ), *pthread_rwlock_clockwrlock*( ), *pthread_rwlock_destroy*( ), *pthread_rwlock_rdlock*( ), *pthread_rwlock_unlock*( )

XBD Section 3.275 (on page 72), Section 4.15.2 (on page 104), **<pthread.h>**

CHANGE HISTORY

First released in Issue 5.

Issue 6

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- The [EDEADLK] error is deleted as a *pthread_rwlock_trywrlock*( ) error.
- The SEE ALSO section is updated.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/104 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

Issue 7

The *pthread_rwlock_trywrlock*( ) and *pthread_rwlock_wrlock*( ) functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results in undefined behavior.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0290 [720] and XSH/TC2-2008/0291 [722] are applied.

59684 **NAME**

59685 pthread_rwlock_unlock — unlock a read-write lock object

59686 **SYNOPSIS**

59687 `#include <pthread.h>`

59688 `int pthread_rwlock_unlock(pthread_rwlock_t *rwlock);`

59689 **DESCRIPTION**

59690 The *pthread_rwlock_unlock*( ) function shall release a lock held on the read-write lock object
59691 referenced by *rwlock*. Results are undefined if the read-write lock *rwlock* is not held by the
59692 calling thread.

59693 If this function is called to release a read lock from the read-write lock object and there are other
59694 read locks currently held on this read-write lock object, the read-write lock object remains in the
59695 read locked state. If this function releases the last read lock for this read-write lock object, the
59696 read-write lock object shall be put in the unlocked state with no owners.

59697 If this function is called to release a write lock for this read-write lock object, the read-write lock
59698 object shall be put in the unlocked state.

59699 If there are threads blocked on the lock when it becomes available, the scheduling policy shall
59700 TPS determine which thread(s) shall acquire the lock. If the Thread Execution Scheduling option is
59701 supported, when threads executing with the scheduling policies SCHED_FIFO, SCHED_RR, or
59702 SCHED_SPORADIC are waiting on the lock, they shall acquire the lock in priority order when
59703 the lock becomes available. For equal priority threads, write locks shall take precedence over
59704 read locks. If the Thread Execution Scheduling option is not supported, it is implementation-
59705 defined whether write locks take precedence over read locks.

59706 Results are undefined if this function is called with an uninitialized read-write lock.

59707 **RETURN VALUE**

59708 If successful, the *pthread_rwlock_unlock*( ) function shall return zero; otherwise, an error number
59709 shall be returned to indicate the error.

59710 **ERRORS**

59711 The *pthread_rwlock_unlock*( ) function shall not return an error code of [EINTR].

59712 **EXAMPLES**

59713 None.

59714 **APPLICATION USAGE**

59715 None.

59716 **RATIONALE**

59717 If an implementation detects that the value specified by the *rwlock* argument to
59718 *pthread_rwlock_unlock*( ) does not refer to an initialized read-write lock object, it is recommended
59719 that the function should fail and report an [EINVAL] error.

59720 If an implementation detects that the value specified by the *rwlock* argument to
59721 *pthread_rwlock_unlock*( ) refers to a read-write lock object for which the current thread does not
59722 hold a lock, it is recommended that the function should fail and report an [EPERM] error.

59723 **FUTURE DIRECTIONS**

59724 None.

SEE ALSO

pthread_rwlock_clockrdlock( ), *pthread_rwlock_clockwrlock*( ), *pthread_rwlock_destroy*( ), *pthread_rwlock_rdlock*( ), *pthread_rwlock_trywrlock*( )

XBD Section 4.15.2 (on page 104), **<pthread.h>**

CHANGE HISTORY

First released in Issue 5.

Issue 6

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- The DESCRIPTION is updated as follows:
  - — The conditions under which writers have precedence over readers are specified.
  - — The concept of read-write lock owner is deleted.
- The SEE ALSO section is updated.

Issue 7

SD5-XSH-ERN-183 is applied.

The *pthread_rwlock_unlock*( ) function is moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results in undefined behavior.

The [EPERM] error for a read-write lock object for which the current thread does not hold a lock is removed; this condition results in undefined behavior.

59747 **NAME**

59748 pthread_rwlock_wrlock — lock a read-write lock object for writing

59749 **SYNOPSIS**

59750 `#include <pthread.h>`

59751 `int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);`

59752 **DESCRIPTION**

59753 Refer to *pthread_rwlock_trywrlock( )*.

NAME

pthread_rwlockattr_destroy, pthread_rwlockattr_init — destroy and initialize the read-write lock attributes object

SYNOPSIS

```
#include <pthread.h>

int pthread_rwlockattr_destroy(pthread_rwlockattr_t *attr);
int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);
```

DESCRIPTION

The *pthread_rwlockattr_destroy*( ) function shall destroy a read-write lock attributes object. A destroyed *attr* attributes object can be reinitialized using *pthread_rwlockattr_init*( ); the results of otherwise referencing the object after it has been destroyed are undefined. An implementation may cause *pthread_rwlockattr_destroy*( ) to set the object referenced by *attr* to an invalid value.

The *pthread_rwlockattr_init*( ) function shall initialize a read-write lock attributes object *attr* with the default value for all of the attributes defined by the implementation.

Results are undefined if *pthread_rwlockattr_init*( ) is called specifying an already initialized *attr* attributes object.

After a read-write lock attributes object has been used to initialize one or more read-write locks, any function affecting the attributes object (including destruction) shall not affect any previously initialized read-write locks.

The behavior is undefined if the value specified by the *attr* argument to *pthread_rwlockattr_destroy*( ) does not refer to an initialized read-write lock attributes object.

RETURN VALUE

If successful, the *pthread_rwlockattr_destroy*( ) and *pthread_rwlockattr_init*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlockattr_init*( ) function shall fail if:

[ENOMEM] Insufficient memory exists to initialize the read-write lock attributes object.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_rwlockattr_destroy*( ) does not refer to an initialized read-write lock attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_destroy( ), *pthread_rwlockattr_getpshared*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5.

Issue 6

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- The SEE ALSO section is updated.

Issue 7

The *pthread_rwlockattr_destroy*( ) and *pthread_rwlockattr_init*( ) functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized read-write lock attributes object is removed; this condition results in undefined behavior.

NAME

pthread_rwlockattr_getpshared, pthread_rwlockattr_setpshared — get and set the process-shared attribute of the read-write lock attributes object

SYNOPSIS

TSH

```
#include <pthread.h>

int pthread_rwlockattr_getpshared(const pthread_rwlockattr_t
    *restrict attr, int *restrict pshared);
int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
    int pshared);
```

DESCRIPTION

The *pthread_rwlockattr_getpshared*( ) function shall obtain the value of the *process-shared* attribute from the initialized attributes object referenced by *attr*. The *pthread_rwlockattr_setpshared*( ) function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

The *process-shared* attribute shall be set to PTHREAD_PROCESS_SHARED to permit a read-write lock to be operated upon by any thread that has access to the memory where the read-write lock is allocated, even if the read-write lock is allocated in memory that is shared by multiple processes. See Section 2.9.9 (on page 548) for further requirements. The default value of the *process-shared* attribute shall be PTHREAD_PROCESS_PRIVATE.

Additional attributes, their default values, and the names of the associated functions to get and set those attribute values are implementation-defined.

The behavior is undefined if the value specified by the *attr* argument to *pthread_rwlockattr_getpshared*( ) or *pthread_rwlockattr_setpshared*( ) does not refer to an initialized read-write lock attributes object.

RETURN VALUE

Upon successful completion, the *pthread_rwlockattr_getpshared*( ) function shall return zero and store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise, an error number shall be returned to indicate the error.

If successful, the *pthread_rwlockattr_setpshared*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlockattr_setpshared*( ) function may fail if:

[EINVAL] The new value specified for the attribute is outside the range of legal values for that attribute.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_destroy( ), *pthread_rwlockattr_destroy*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5.

Issue 6

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR TSH to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- The DESCRIPTION notes that additional attributes are implementation-defined.
- The SEE ALSO section is updated.

The **restrict** keyword is added to the *pthread_rwlockattr_getpshared*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

The *pthread_rwlockattr_getpshared*( ) and *pthread_rwlockattr_setpshared*( ) functions are marked only as part of the Thread Process-Shared Synchronization option as the Threads option is now part of the Base.

The [EINVAL] error for an uninitialized read-write lock attributes object is removed; this condition results in undefined behavior.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0292 [972] and XSH/TC2-2008/0293 [757] are applied.

59872 **NAME**

59873 pthread_rwlockattr_init — initialize the read-write lock attributes object

59874 **SYNOPSIS**

59875 `#include <pthread.h>`

59876 `int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);`

59877 **DESCRIPTION**

59878 Refer to *pthread_rwlockattr_destroy( )*.

NAME

pthread_rwlockattr_setpshared — set the process-shared attribute of the read-write lock attributes object

SYNOPSIS

TSH

```
#include <pthread.h>

int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
    int pshared);
```

DESCRIPTION

Refer to *pthread_rwlockattr_getpshared( )*.

NAME

pthread_self — get the calling thread ID

SYNOPSIS

```
#include <pthread.h>

pthread_t pthread_self(void);
```

DESCRIPTION

The *pthread_self*( ) function shall return the thread ID of the calling thread.

RETURN VALUE

The *pthread_self*( ) function shall always be successful and no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *pthread_self*( ) function provides a capability similar to the *getpid*( ) function for processes and the rationale is the same: the creation call does not provide the thread ID to the created thread.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_create( ), *pthread_equal*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_self*( ) function is marked as part of the Threads option.

Issue 7

Austin Group Interpretation 1003.1-2001 #063 is applied, updating the RETURN VALUE section.

The *pthread_self*( ) function is moved from the Threads option to the Base.

NAME

pthread_setcancelstate, pthread_setcanceltype, pthread_testcancel — set cancelability state

SYNOPSIS

```
#include <pthread.h>

int pthread_setcancelstate(int state, int *oldstate);
int pthread_setcanceltype(int type, int *oldtype);
void pthread_testcancel(void);
```

DESCRIPTION

The *pthread_setcancelstate*( ) function shall atomically both set the calling thread's cancelability state to the indicated *state* and return the previous cancelability state at the location referenced by *oldstate*. Legal values for *state* are PTHREAD_CANCEL_ENABLE and PTHREAD_CANCEL_DISABLE.

The *pthread_setcanceltype*( ) function shall atomically both set the calling thread's cancelability type to the indicated *type* and return the previous cancelability type at the location referenced by *oldtype*. Legal values for *type* are PTHREAD_CANCEL_DEFERRED and PTHREAD_CANCEL_ASYNCHRONOUS.

The cancelability state and type of any newly created threads, including the thread in which *main*( ) was first invoked, shall be PTHREAD_CANCEL_ENABLE and PTHREAD_CANCEL_DEFERRED respectively.

The *pthread_testcancel*( ) function shall create a cancellation point in the calling thread. The *pthread_testcancel*( ) function shall have no effect if cancelability is disabled.

The *pthread_setcancelstate*( ) function shall be async-signal-safe.

RETURN VALUE

If successful, the *pthread_setcancelstate*( ) and *pthread_setcanceltype*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_setcancelstate*( ) function may fail if:

[EINVAL] The specified state is not PTHREAD_CANCEL_ENABLE or PTHREAD_CANCEL_DISABLE.

The *pthread_setcanceltype*( ) function may fail if:

[EINVAL] The specified type is not PTHREAD_CANCEL_DEFERRED or PTHREAD_CANCEL_ASYNCHRONOUS.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

In order to write a signal handler for an asynchronous signal which can run safely in a cancellable thread, *pthread_setcancelstate*( ) must be used to disable cancellation for the duration of any calls that the signal handler makes which are cancellation points. However, earlier versions of the standard did not permit strictly conforming applications to call *pthread_setcancelstate*( ) from a signal handler since it was not required to be async-signal-safe. On non-conforming implementations where *pthread_setcancelstate*( ) is not async-signal-safe, alternatives are to ensure either that the corresponding signals are blocked during execution of functions that are not async-cancel-safe or that cancellation is disabled during times when those signals could be delivered.

RATIONALE

The *pthread_setcancelstate*( ) and *pthread_setcanceltype*( ) functions control the points at which a thread may be asynchronously canceled. For cancellation control to be usable in modular fashion, some rules need to be followed.

An object can be considered to be a generalization of a procedure. It is a set of procedures and global variables written as a unit and called by clients not known by the object. Objects may depend on other objects.

First, cancelability should only be disabled on entry to an object, never explicitly enabled. On exit from an object, the cancelability state should always be restored to its value on entry to the object.

This follows from a modularity argument: if the client of an object (or the client of an object that uses that object) has disabled cancelability, it is because the client does not want to be concerned about cleaning up if the thread is canceled while executing some sequence of actions. If an object is called in such a state and it enables cancelability and a cancellation request is pending for that thread, then the thread is canceled, contrary to the wish of the client that disabled.

Second, the cancelability type may be explicitly set to either *deferred* or *asynchronous* upon entry to an object. But as with the cancelability state, on exit from an object the cancelability type should always be restored to its value on entry to the object.

Finally, only functions that are cancel-safe may be called from a thread that is asynchronously cancelable.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cancel( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_setcancelstate*( ), *pthread_setcanceltype*( ), and *pthread_testcancel*( ) functions are marked as part of the Threads option.

Issue 7

The *pthread_setcancelstate*( ), *pthread_setcanceltype*( ), and *pthread_testcancel*( ) functions are moved from the Threads option to the Base.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0294 [622] and XSH/TC2-2008/0295 [615] are applied.

Issue 8

Austin Group Defect 841 is applied, requiring *pthread_setcancelstate*( ) to be async-signal-safe.

NAME

pthread_setschedparam — dynamic thread scheduling parameters access (**REALTIME THREADS**)

SYNOPSIS

TPS

```
#include <pthread.h>

int pthread_setschedparam(pthread_t thread, int policy,
    const struct sched_param *param);
```

DESCRIPTION

Refer to *pthread_getschedparam( )*.

NAME

pthread_setschedprio — dynamic thread scheduling parameters access (**REALTIME THREADS**)

SYNOPSIS

TPS

```
#include <pthread.h>

int pthread_setschedprio(pthread_t thread, int prio);
```

DESCRIPTION

The *pthread_setschedprio*( ) function shall set the scheduling priority for the thread whose thread ID is given by *thread* to the value given by *prio*. See Scheduling Policies (on page 531) for a description on how this function call affects the ordering of the thread in the thread list for its new priority.

If the *pthread_setschedprio*( ) function fails, the scheduling priority of the target thread shall not be changed.

RETURN VALUE

If successful, the *pthread_setschedprio*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_setschedprio*( ) function may fail if:

[EINVAL] The value of *prio* is invalid for the scheduling policy of the specified thread.

[EPERM] The caller does not have appropriate privileges to set the scheduling priority of the specified thread.

The *pthread_setschedprio*( ) function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *pthread_setschedprio*( ) function provides a way for an application to temporarily raise its priority and then lower it again, without having the undesired side-effect of yielding to other threads of the same priority. This is necessary if the application is to implement its own strategies for bounding priority inversion, such as priority inheritance or priority ceilings. This capability is especially important if the implementation does not support the Thread Priority Protection or Thread Priority Inheritance options, but even if those options are supported it is needed if the application is to bound priority inheritance for other resources, such as semaphores.

The standard developers considered that while it might be preferable conceptually to solve this problem by modifying the specification of *pthread_setschedparam*( ), it was too late to make such a change, as there may be implementations that would need to be changed. Therefore, this new function was introduced.

If an implementation detects use of a thread ID after the end of its lifetime, it is recommended that the function should fail and report an [ESRCH] error.

60052 **FUTURE DIRECTIONS**

60053 None.

60054 **SEE ALSO**

60055 Scheduling Policies (on page 531), *pthread_getschedparam*( )

60056 XBD **<pthread.h>**

60057 **CHANGE HISTORY**

60058 First released in Issue 6. Included as a response to IEEE PASC Interpretation 1003.1 #96.

60059 **Issue 7**

60060 The *pthread_setschedprio*( ) function is marked only as part of the Thread Execution Scheduling
60061 option as the Threads option is now part of the Base.

60062 Austin Group Interpretation 1003.1-2001 #069 is applied, updating the [EPERM] error.

60063 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

60064 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0466 [314] is applied.

60065 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0296 [757] is applied.

60066 **NAME**

60067 pthread_setspecific — thread-specific data management

60068 **SYNOPSIS**

60069
```
#include <pthread.h>
```

60070
```
int pthread_setspecific(pthread_key_t key, const void *value);
```

60071 **DESCRIPTION**

60072 Refer to *pthread_getspecific( )*.

60073 **NAME**

60074 pthread_sigmask, sigprocmask — examine and change blocked signals

60075 **SYNOPSIS**

60076 CX
```
#include <signal.h>

int pthread_sigmask(int how, const sigset_t *restrict set,
    sigset_t *restrict oset);
int sigprocmask(int how, const sigset_t *restrict set,
    sigset_t *restrict oset);
```

60081 **DESCRIPTION**

60082 The *pthread_sigmask*( ) function shall examine or change (or both) the calling thread's signal
60083 mask.

60084 If the argument *set* is not a null pointer, it points to a set of signals to be used to change the
60085 currently blocked set.

60086 The argument *how* indicates the way in which the set is changed, and the application shall
60087 ensure it consists of one of the following values:

60088 SIG_BLOCK The resulting set shall be the union of the current set and the signal set
60089 pointed to by *set*.

60090 SIG_SETMASK The resulting set shall be the signal set pointed to by *set*.

60091 SIG_UNBLOCK The resulting set shall be the intersection of the current set and the
60092 complement of the signal set pointed to by *set*.

60093 If the argument *oset* is not a null pointer, the previous mask shall be stored in the location
60094 pointed to by *oset*. If *set* is a null pointer, the value of the argument *how* is not significant and the
60095 thread's signal mask shall be unchanged; thus the call can be used to enquire about currently
60096 blocked signals.

60097 If the argument *set* is not a null pointer, after *pthread_sigmask*( ) changes the currently blocked set
60098 of signals it shall determine whether there are any pending unblocked signals; if there are any,
60099 then at least one of those signals shall be delivered before the call to *pthread_sigmask*( ) returns.

60100 It is not possible to block those signals which cannot be ignored. This shall be enforced by the
60101 system without causing an error to be indicated.

60102 If any of the SIGFPE, SIGILL, SIGSEGV, or SIGBUS signals are generated while they are blocked,
60103 the result is undefined, unless the signal was generated by the action of another process, or by
60104 one of the functions *kill*( ), *pthread_kill*( ), *raise*( ), or *sigqueue*( ).

60105 If *pthread_sigmask*( ) fails, the thread's signal mask shall not be changed.

60106 The *sigprocmask*( ) function shall be equivalent to *pthread_sigmask*( ), except that its behavior is
60107 unspecified if called from a multi-threaded process, and on error it returns –1 and sets *errno* to
60108 the error number instead of returning the error number directly.

60109 **RETURN VALUE**

60110 Upon successful completion, *pthread_sigmask*( ) shall return 0; otherwise, it shall return the
60111 corresponding error number.

60112 Upon successful completion, *sigprocmask*( ) shall return 0; otherwise, –1 shall be returned and
60113 *errno* shall be set to indicate the error.

ERRORS

These functions shall fail if:

[EINVAL] The *set* argument is not a null pointer and the value of the *how* argument is not equal to one of the defined values.

These functions shall not return an error code of [EINTR].

EXAMPLES

Signaling in a Multi-Threaded Process

This example shows the use of *pthread_sigmask*( ) in order to deal with signals in a multi-threaded process. It provides a fairly general framework that could be easily adapted/extended.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <signal.h>
#include <string.h>
#include <errno.h>
...

static sigset_t   signal_mask;  /* signals to block         */

int main (int argc, char *argv[])
{
    pthread_t  sig_thr_id;      /* signal handler thread ID */
    int        rc;              /* return code              */

    sigemptyset (&signal_mask);
    sigaddset (&signal_mask, SIGINT);
    sigaddset (&signal_mask, SIGTERM);
    rc = pthread_sigmask (SIG_BLOCK, &signal_mask, NULL);
    if (rc != 0) {
        /* handle error */
        ...
    }
    /* any newly created threads inherit the signal mask */

    rc = pthread_create (&sig_thr_id, NULL, signal_thread, NULL);
    if (rc != 0) {
        /* handle error */
        ...
    }

    /* APPLICATION CODE */
    ...
}

void *signal_thread (void *arg)
{
    int       sig_caught;    /* signal caught       */
    int       rc;            /* returned code       */

    rc = sigwait (&signal_mask, &sig_caught);
    if (rc != 0) {
        /* handle error */
```



```
60159           }
60160           switch (sig_caught)
60161           {
60162           case SIGINT:     /* process SIGINT  */
60163               ...
60164               break;
60165           case SIGTERM:    /* process SIGTERM */
60166               ...
60167               break;
60168           default:         /* should normally not happen */
60169               fprintf (stderr, "\nUnexpected signal %d\n", sig_caught);
60170               break;
60171           }
60172       }
```

APPLICATION USAGE

Although *pthread_sigmask*( ) has to deliver at least one of any pending unblocked signals that exist after it has changed the currently blocked set of signals, there is no requirement that the delivered signal(s) include any that were unblocked by the change. If one or more signals that were already unblocked become pending (see Section 2.4.1, on page 513) during the period the *pthread_setmask*( ) call is executing, the signal(s) delivered before the call returns might include only those signals.

RATIONALE

When a thread's signal mask is changed in a signal-catching function that is installed by *sigaction*( ), the restoration of the signal mask on return from the signal-catching function overrides that change (see *sigaction*( )). If the signal-catching function was installed with *signal*( ), it is unspecified whether this occurs.

See *kill*( ) for a discussion of the requirement on delivery of signals.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *kill*( ), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigemptyset*( ), *sigfillset*( ), *sigismember*( ), *sigpending*( ), *sigqueue*( ), *sigsuspend*( )

XBD **<signal.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

The *pthread_sigmask*( ) function is added for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_sigmask*( ) function is marked as part of the Threads option.

The SYNOPSIS for *sigprocmask*( ) is marked as a CX extension to note that the presence of this function in the **<signal.h>** header is an extension to the ISO C standard.

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- The DESCRIPTION is updated to explicitly state the functions which may generate the signal.

The normative text is updated to avoid use of the term ``must'' for application requirements.

The **restrict** keyword is added to the *pthread_sigmask*( ) and *sigprocmask*( ) prototypes for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/105 is applied, updating ``process' signal mask'' to ``thread's signal mask'' in the DESCRIPTION and RATIONALE sections.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/106 is applied, adding the example to the EXAMPLES section.

Issue 7

The *pthread_sigmask*( ) function is moved from the Threads option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0467 [319] is applied.

Issue 8

Austin Group Defect 1132 is applied, clarifying the [EINVAL] error.

Austin Group Defect 1636 is applied, clarifying the exceptions to the equivalence of *pthread_sigmask*( ) and *sigprocmask*( ).

Austin Group Defect 1731 is applied, clarifying that although *pthread_sigmask*( ) has to deliver at least one of any pending unblocked signals that exist after it has changed the currently blocked set of signals, there is no requirement that the delivered signal(s) include any that were unblocked by the change.

60222 **NAME**

60223 pthread_spin_destroy, pthread_spin_init — destroy or initialize a spin lock object

60224 **SYNOPSIS**

60225 ```
#include <pthread.h>

int pthread_spin_destroy(pthread_spinlock_t *lock);
int pthread_spin_init(pthread_spinlock_t *lock, int pshared);
```

60226

60227

60228 **DESCRIPTION**

60229 The *pthread_spin_destroy*( ) function shall destroy the spin lock referenced by *lock* and release any
60230 resources used by the lock. The effect of subsequent use of the lock is undefined until the lock is
60231 reinitialized by another call to *pthread_spin_init*( ). The results are undefined if
60232 *pthread_spin_destroy*( ) is called when a thread holds the lock, or if this function is called with an
60233 uninitialized thread spin lock.

60234 The *pthread_spin_init*( ) function shall allocate any resources required to use the spin lock
60235 referenced by *lock* and initialize the lock to an unlocked state.

60236 TSH If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is
60237 PTHREAD_PROCESS_SHARED, the implementation shall permit the spin lock to be operated
60238 upon by any thread that has access to the memory where the spin lock is allocated, even if it is
60239 allocated in memory that is shared by multiple processes.

60240 See Section 2.9.9 (on page 548) for further requirements.

60241 The results are undefined if *pthread_spin_init*( ) is called specifying an already initialized spin
60242 lock. The results are undefined if a spin lock is used without first being initialized.

60243 If the *pthread_spin_init*( ) function fails, the lock is not initialized and the contents of *lock* are
60244 undefined.

60245 Only the object referenced by *lock* may be used for performing synchronization.

60246 The result of referring to copies of that object in calls to *pthread_spin_destroy*( ),
60247 *pthread_spin_lock*( ), *pthread_spin_trylock*( ), or *pthread_spin_unlock*( ) is undefined.

60248 **RETURN VALUE**

60249 Upon successful completion, these functions shall return zero; otherwise, an error number shall
60250 be returned to indicate the error.

60251 **ERRORS**

60252 The *pthread_spin_init*( ) function shall fail if:

60253 [EAGAIN] The system lacks the necessary resources to initialize another spin lock.

60254 [ENOMEM] Insufficient memory exists to initialize the lock.

60255 These functions shall not return an error code of [EINTR].

60256 **EXAMPLES**

60257 None.

60258 **APPLICATION USAGE**

60259 None.

60260 **RATIONALE**

60261 If an implementation detects that the value specified by the *lock* argument to
60262 *pthread_spin_destroy*( ) does not refer to an initialized spin lock object, it is recommended that the
60263 function should fail and report an [EINVAL] error.

60264 If an implementation detects that the value specified by the *lock* argument to

  
```

pthread_spin_destroy( ) or *pthread_spin_init*( ) refers to a locked spin lock object, or detects that the value specified by the *lock* argument to *pthread_spin_init*( ) refers to an already initialized spin lock object, it is recommended that the function should fail and report an [EBUSY] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_spin_lock( ), *pthread_spin_unlock*( )

XBD **<pthread.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

Issue 7

The *pthread_spin_destroy*( ) and *pthread_spin_init*( ) functions are moved from the Spin Locks option to the Base.

The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in undefined behavior.

The [EBUSY] error for a locked spin lock object or an already initialized spin lock object is removed; this condition results in undefined behavior.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0297 [972] is applied.

NAME

pthread_spin_lock, pthread_spin_trylock — lock a spin lock object

SYNOPSIS

```
#include <pthread.h>

int pthread_spin_lock(pthread_spinlock_t *lock);
int pthread_spin_trylock(pthread_spinlock_t *lock);
```

DESCRIPTION

The *pthread_spin_lock*( ) function shall lock the spin lock referenced by *lock*. The calling thread shall acquire the lock if it is not held by another thread. Otherwise, the thread shall spin (that is, shall not return from the *pthread_spin_lock*( ) call) until the lock becomes available. The results are undefined if the calling thread holds the lock at the time the call is made. The *pthread_spin_trylock*( ) function shall lock the spin lock referenced by *lock* if it is not held by any thread. Otherwise, the function shall fail.

The results are undefined if any of these functions is called with an uninitialized spin lock.

RETURN VALUE

Upon successful completion, these functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_spin_lock*( ) function may fail if:

[EDEADLK] A deadlock condition was detected.

The *pthread_spin_trylock*( ) function shall fail if:

[EBUSY] A thread currently holds the lock.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications using this function may be subject to priority inversion, as discussed in XBD Section 3.275 (on page 72).

RATIONALE

If an implementation detects that the value specified by the *lock* argument to *pthread_spin_lock*( ) or *pthread_spin_trylock*( ) does not refer to an initialized spin lock object, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the *lock* argument to *pthread_spin_lock*( ) refers to a spin lock object for which the calling thread already holds the lock, it is recommended that the function should fail and report an [EDEADLK] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_spin_destroy( ), *pthread_spin_unlock*( )

XBD Section 3.275 (on page 72), Section 4.15.2 (on page 104), **<pthread.h>**

60324 **CHANGE HISTORY**

60325 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

60326 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

60327 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/107 is applied, updating the ERRORS
60328 section so that the [EDEADLK] error includes detection of a deadlock condition.

60329 **Issue 7**

60330 The *pthread_spin_lock*( ) and *pthread_spin_trylock*( ) functions are moved from the Spin Locks
60331 option to the Base.

60332 The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in
60333 undefined behavior.

60334 The [EDEADLK] error for a spin lock object for which the calling thread already holds the lock is
60335 removed; this condition results in undefined behavior.

60336 **NAME**

60337 pthread_spin_unlock — unlock a spin lock object

60338 **SYNOPSIS**

60339 ```
#include <pthread.h>
```

60340 ```
int pthread_spin_unlock(pthread_spinlock_t *lock);
```

60341 **DESCRIPTION**

60342 The *pthread_spin_unlock*( ) function shall release the spin lock referenced by *lock* which was
60343 locked via the *pthread_spin_lock*( ) or *pthread_spin_trylock*( ) functions.

60344 The results are undefined if the lock is not held by the calling thread.

60345 If there are threads spinning on the lock when *pthread_spin_unlock*( ) is called, the lock becomes
60346 available and an unspecified spinning thread shall acquire the lock.

60347 The results are undefined if this function is called with an uninitialized thread spin lock.

60348 **RETURN VALUE**

60349 Upon successful completion, the *pthread_spin_unlock*( ) function shall return zero; otherwise, an
60350 error number shall be returned to indicate the error.

60351 **ERRORS**

60352 This function shall not return an error code of [EINTR].

60353 **EXAMPLES**

60354 None.

60355 **APPLICATION USAGE**

60356 None.

60357 **RATIONALE**

60358 If an implementation detects that the value specified by the *lock* argument to
60359 *pthread_spin_unlock*( ) does not refer to an initialized spin lock object, it is recommended that the
60360 function should fail and report an [EINVAL] error.

60361 If an implementation detects that the value specified by the *lock* argument to
60362 *pthread_spin_unlock*( ) refers to a spin lock object for which the current thread does not hold the
60363 lock, it is recommended that the function should fail and report an [EPERM] error.

60364 **FUTURE DIRECTIONS**

60365 None.

60366 **SEE ALSO**

60367 *pthread_spin_destroy*( ), *pthread_spin_lock*( )

60368 XBD Section 4.15.2 (on page 104), **<pthread.h>**

60369 **CHANGE HISTORY**

60370 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

60371 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

60372 **Issue 7**

60373 The *pthread_spin_unlock*( ) function is moved from the Spin Locks option to the Base.

60374 The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in
60375 undefined behavior.

60376 The [EPERM] error for a spin lock object for which the current thread does not hold the lock is
60377 removed; this condition results in undefined behavior.

60378 **NAME**

60379 pthread_testcancel — set cancelability state

60380 **SYNOPSIS**

60381 `#include <pthread.h>`

60382 `void pthread_testcancel(void);`

60383 **DESCRIPTION**

60384 Refer to *pthread_setcancelstate( )*.

NAME

ptsname, ptsname_r — get name of the subsidiary pseudo-terminal device

SYNOPSIS

XSI
```
#include <stdlib.h>

char *ptsname(int fildes);
int ptsname_r(int fildes, char *name, size_t namesize);
```

DESCRIPTION

The *ptsname*( ) function shall return the name of the subsidiary pseudo-terminal device associated with a manager pseudo-terminal device. The *fildes* argument is a file descriptor that refers to the manager device. The *ptsname*( ) function shall return a pointer to a string containing the pathname of the corresponding subsidiary device.

The *ptsname*( ) function need not be thread-safe.

The *ptsname_r*( ) function shall store the name of the subsidiary pseudo-terminal device corresponding to *fildes* in the character array referenced by *name*. The array is *namesize* characters long and should have space for the name and the terminating null character. The maximum length of the terminal name shall be {TTY_NAME_MAX}.

RETURN VALUE

Upon successful completion, *ptsname*( ) shall return a pointer to a string which is the name of the pseudo-terminal subsidiary device. Upon failure, *ptsname*( ) shall return a null pointer and may set *errno*. This could occur if *fildes* is an invalid file descriptor or if the subsidiary device name does not exist in the file system.

The application shall not modify the string returned. The returned pointer might be invalidated or the string content might be overwritten by a subsequent call to *ptsname*( ). The returned pointer and the string content might also be invalidated if the calling thread is terminated.

If successful, the *ptsname_r*( ) function shall return zero. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *ptsname_r*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

[EINVAL] The *name* argument is a null pointer.

[ERANGE] The value of *namesize* is smaller than the length of the string to be returned including the terminating null character.

The *ptsname*( ) function may fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

The *ptsname*( ) and *ptsname_r*( ) functions may fail if:

[EINVAL] or [ENOTTY]
The file associated with the *fildes* argument is not a manager pseudo-terminal device.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *ptsname_r*( ) function is required to make it possible for a multi-threaded program to safely determine the name of a subsidiary device. Although the name of the device is constrained by {TTY_NAME_MAX}, this value might not be a compile-time constant, so an application can rely on repeated calls with successively larger buffers until the result is no longer [ERANGE] as an alternative for properly sizing the buffer.

Historically, some versions of *ptsname*( ) did not set *errno* even when returning a null pointer. However, *ptsname_r*( ) is required to either populate the buffer with a valid name or return an error value.

See also the RATIONALE section for *posix_openpt*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

grantpt( ), *open*( ), *posix_openpt*( ), *ttyname*( ), *unlockpt*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0468 [75] and XSH/TC1-2008/0469 [96] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0298 [503], XSH/TC2-2008/0299 [656], and XSH/TC2-2008/0300 [503] are applied.

Issue 8

Austin Group Defect 508 is applied, adding the *ptsname_r*( ) function.

Austin Group Defect 1466 is applied, changing the terminology used for pseudo-terminal devices.

60457 **NAME**

60458 putc — put a byte on a stream

60459 **SYNOPSIS**

60460 `#include <stdio.h>`

60461 `int putc(int c, FILE *stream);`

60462 **DESCRIPTION**

60463 CX The functionality described on this reference page is aligned with the ISO C standard. Any
60464 conflict between the requirements described here and the ISO C standard is unintentional. This
60465 volume of POSIX.1-2024 defers to the ISO C standard.

60466 The *putc*( ) function shall be equivalent to *fputc*( ), except that if it is implemented as a macro it
60467 may evaluate *stream* more than once, so the argument should never be an expression with side-
60468 effects.

60469 **RETURN VALUE**

60470 Refer to *fputc*( ).

60471 **ERRORS**

60472 Refer to *fputc*( ).

60473 **EXAMPLES**

60474 None.

60475 **APPLICATION USAGE**

60476 Since it may be implemented as a macro, *putc*( ) may treat a *stream* argument with side-effects
60477 incorrectly. In particular, *putc*(*c*,*f*++) does not necessarily work correctly. Therefore, use of this
60478 function is not recommended in such situations; *fputc*( ) should be used instead.

60479 **RATIONALE**

60480 None.

60481 **FUTURE DIRECTIONS**

60482 None.

60483 **SEE ALSO**

60484 Section 2.5 (on page 521), *fputc*( )

60485 XBD **<stdio.h>**

60486 **CHANGE HISTORY**

60487 First released in Issue 1. Derived from Issue 1 of the SVID.

60488 **Issue 7**

60489 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0470 [14] is applied.

60490 **NAME**

60491 putc_unlocked — stdio with explicit client locking

60492 **SYNOPSIS**

60493 CX
```
#include <stdio.h>

int putc_unlocked(int c, FILE *stream);
```

60495 **DESCRIPTION**

60496 Refer to *getc_unlocked*( ).

NAME

putchar — put a byte on a stdout stream

SYNOPSIS

```
#include <stdio.h>

int putchar(int c);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The function call *putchar*(*c*) shall be equivalent to *putc*(*c*,*stdout*).

RETURN VALUE

Refer to *fputc( )*.

ERRORS

Refer to *fputc( )*.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *putc( )*

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0471 [14] is applied.

NAME

putchar_unlocked — stdio with explicit client locking

SYNOPSIS

CX
```
#include <stdio.h>

int putchar_unlocked(int c);
```

DESCRIPTION

Refer to *getc_unlocked*( ).

NAME

putenv — change or add a value to an environment

SYNOPSIS

XSI
```
#include <stdlib.h>

int putenv(char *string);
```

DESCRIPTION

The *putenv*( ) function shall use the *string* argument to set, or optionally unset, an environment variable value:

- If the *string* argument points to a string of the form "*name=value*", where *name* is a valid name, the *putenv*( ) function shall make the value of the environment variable with that name equal to *value* by altering an existing variable or creating a new one. In either case, the string pointed to by *string* shall become part of the environment, so altering the string shall change the environment.
- If the *string* argument points to a string containing a valid name, the *putenv*( ) function shall either remove the environment variable with that name (if it exists) from the environment or fail with *errno* set to [EINVAL].
- Otherwise, the behavior is unspecified.

The *putenv*( ) function need not be thread-safe.

RETURN VALUE

Upon successful completion, *putenv*( ) shall return 0; otherwise, it shall return a non-zero value and set *errno* to indicate the error.

ERRORS

The *putenv*( ) function may fail if:

[EINVAL] The *string* argument points to a string that is not of the form "*name=value*", where *name* is a valid name.

[ENOMEM] Insufficient memory was available.

EXAMPLES

Changing the Value of an Environment Variable

The following example changes the value of the *HOME* environment variable to the value **/usr/home**.

```
#include <stdlib.h>
...
static char *var = "HOME=/usr/home";
int ret;

ret = putenv(var);
```

APPLICATION USAGE

The *putenv*( ) function manipulates the environment pointed to by *environ*, and can be used in conjunction with *getenv*( ).

See *exec*( ) for restrictions on changing the environment in multi-threaded applications.

This routine may use *malloc*( ) to enlarge the environment.

A potential error is to call *putenv*( ) with an automatic variable as the argument, then return from

the calling function while *string* is still part of the environment.

Although the space used by *string* is no longer used once a new string which defines *name* is passed to *putenv*( ), if any thread in the application has used *getenv*( ) to retrieve a pointer to this variable, it should not be freed by calling *free*( ). If the changed environment variable is one known by the system (such as the locale environment variables) the application should never free the buffer used by earlier calls to *putenv*( ) for the same variable.

The *setenv*( ) function is preferred over this function. One reason is that *putenv*( ) is optional and therefore less portable. Another is that using *putenv*( ) can slow down environment searches, as explained in the RATIONALE section for *getenv*( ).

RATIONALE

Refer to the RATIONALE section in *setenv*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *free*( ), *getenv*( ), *malloc*( ), *setenv*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The type of the argument to this function is changed from **const char** * to **char** *. This was indicated as a FUTURE DIRECTION in previous issues.

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/48 is applied, clarifying wording in the DESCRIPTION and adding a new paragraph into APPLICATION USAGE referring readers to *exec*.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0472 [167], XSH/TC1-2008/0473 [167], XSH/TC1-2008/0474 [273,438], and XSH/TC1-2008/0475 [273] are applied.

Issue 8

Austin Group Defect 1598 is applied, specifying the allowed behaviors when the *string* argument points to a string containing a valid name.

NAME

puts — put a string on standard output

SYNOPSIS

```
#include <stdio.h>

int puts(const char *s);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *puts*( ) function shall write the string pointed to by *s*, followed by a <newline>, to the standard output stream *stdout*. The terminating null byte shall not be written.

CX The last data modification and last file status change timestamps of the file shall be marked for update between the successful execution of *puts*( ) and the next successful completion of a call to *fflush*( ) or *fclose*( ) on the same stream or a call to *exit*( ) or *abort*( ).

RETURN VALUE

Upon successful completion, *puts*( ) shall return a non-negative number. Otherwise, it shall return EOF, shall set an error indicator for the stream, CX and *errno* shall be set to indicate the error.

ERRORS

Refer to *fputc*( ).

EXAMPLES

Printing to Standard Output

The following example gets the current time, converts it to a string using *localtime*( ) and *asctime*( ), and prints it to standard output using *puts*( ). It then prints the number of minutes to an event for which it is waiting.

```
#include <time.h>
#include <stdio.h>
...
time_t now;
int minutes_to_event;
...
time(&now);
printf("The time is ");
puts(asctime(localtime(&now)));
printf("There are %d minutes to the event.\n",
    minutes_to_event);
...
```

APPLICATION USAGE

The *puts*( ) function appends a <newline>, while *fputs*( ) does not.

This volume of POSIX.1-2024 requires that successful completion simply return a non-negative integer. There are at least three known different implementation conventions for this requirement:

- Return a constant value.

- Return the last character written.
- Return the number of bytes written. Note that this implementation convention cannot be adhered to for strings longer than {INT_MAX} bytes as the value would not be representable in the return type of the function. For backwards compatibility, implementations can return the number of bytes for strings of up to {INT_MAX} bytes, and return {INT_MAX} for all longer strings.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fopen( )*, *fputs( )*, *putc( )*

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

Issue 7

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0476 [174,412] and XSH/TC1-2008/0477 [14] are applied.

60670 **NAME**

60671 pututxline — put an entry into the user accounting database

60672 **SYNOPSIS**

60673 XSI
```
#include <utmpx.h>

struct utmpx *pututxline(const struct utmpx *utmpx);
```

60674

60675 **DESCRIPTION**

60676 Refer to *endutxent( )*.

NAME

putwc — put a wide character on a stream

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

wint_t putwc(wchar_t wc, FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *putwc*( ) function shall be equivalent to *fputwc*( ), except that if it is implemented as a macro it may evaluate *stream* more than once, so the argument should never be an expression with side-effects.

RETURN VALUE

Refer to *fputwc*( ).

ERRORS

Refer to *fputwc*( ).

EXAMPLES

None.

APPLICATION USAGE

Since it may be implemented as a macro, *putwc*( ) may treat a *stream* argument with side-effects incorrectly. In particular, *putwc*(*wc*,*f*++) need not work correctly. Therefore, use of this function is not recommended; *fputwc*( ) should be used instead.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fputwc*( )

XBD **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released as a World-wide Portability Interface in Issue 4.

Issue 5

Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc* is changed from **wint_t** to **wchar_t**.

The Optional Header (OH) marking is removed from **<stdio.h>**.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0478 [14] is applied.

NAME

putwchar — put a wide character on a stdout stream

SYNOPSIS

```
#include <wchar.h>

wint_t putwchar(wchar_t wc);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The function call *putwchar*(*wc*) shall be equivalent to *putwc*(*wc*,*stdout*).

RETURN VALUE

Refer to *fputwc*( ).

ERRORS

Refer to *fputwc*( ).

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fputwc*( ), *putwc*( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc* is changed from **wint_t** to **wchar_t**.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0479 [14] is applied.

60747 **NAME**

60748 pwrite — write on a file

60749 **SYNOPSIS**

60750 ```
#include <unistd.h>
```

60751 ```
ssize_t pwrite(int fildes, const void *buf, size_t nbyte,
```

60752 ```
    off_t offset);
```

60753 **DESCRIPTION**

60754 Refer to *write( )*.

  
```

NAME

qsort, qsort_r — sort a table of data

SYNOPSIS

```
#include <stdlib.h>

void qsort(void *base, size_t nel, size_t width,
    int (*compar)(const void *, const void *));
CX void qsort_r(void *base, size_t nel, size_t width,
    int (*compar)(const void *, const void *, void *), void *arg);
```

DESCRIPTION

CX For *qsort*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *qsort*( ) function shall sort an array of *nel* objects, the initial element of which is pointed to by *base*. The size of each object, in bytes, is specified by the *width* argument. If the *nel* argument has the value zero, the comparison function pointed to by *compar* shall not be called and no rearrangement shall take place.

The application shall ensure that the comparison function pointed to by *compar* does not alter the contents of the array. The implementation may reorder elements of the array between calls to the comparison function, but shall not alter the contents of any individual element.

When the same objects (consisting of width bytes, irrespective of their current positions in the array) are passed more than once to the comparison function, the results shall be consistent with one another. That is, they shall define a total ordering on the array.

The contents of the array shall be sorted in ascending order according to a comparison function. The *compar* argument is a pointer to the comparison function, which is called with two arguments that point to the elements being compared. The application shall ensure that the function returns an integer less than, equal to, or greater than 0, if the first argument is considered respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is unspecified.

CX The *qsort_r*( ) function shall be identical to *qsort*( ) except that the comparison function *compar* takes a third argument. The *arg* opaque pointer passed to *qsort_r*( ) shall in turn be passed as the third argument to the comparison function.

RETURN VALUE

These functions shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

If the *compar* callback function requires any additional state outside of the items being sorted, it can only access this state through global variables, making it potentially unsafe to use *qsort*( ) with the same *compar* function from separate threads at the same time. The *qsort_r*( ) function was added with the ability to pass through arbitrary arguments to the comparator, which avoids the need to access global variables and thus making it possible to safely share a stateful

comparator across threads.

RATIONALE

The requirement that each argument (hereafter referred to as *p)* to the comparison function is a pointer to elements of the array implies that for every call, for each argument separately, all of the following expressions are non-zero:

```
((char *)p - (char *)base) % width == 0
(char *)p >= (char *)base
(char *)p < (char *)base + nel * width
```

FUTURE DIRECTIONS

None.

SEE ALSO

alphasort( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/49 is applied, adding the last sentence to the first non-shaded paragraph in the DESCRIPTION, and the following two paragraphs. The RATIONALE is also updated. These changes are for alignment with the ISO C standard.

Issue 8

Austin Group Defect 900 is applied, adding the *qsort_r*( ) function.

NAME

quick_exit — terminate a process

SYNOPSIS

```
#include <stdlib.h>

_Noreturn void quick_exit(int status);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *quick_exit*( ) function shall cause normal process termination to occur. It shall not call functions registered with *atexit*( ) nor any registered signal handlers. If a process calls the *quick_exit*( ) function more than once, or calls the *exit*( ) function in addition to the *quick_exit*( ) function, the behavior is undefined. If a signal is raised while the *quick_exit*( ) function is executing, the behavior is undefined.

The *quick_exit*( ) function shall first call all functions registered by *at_quick_exit*( ), in the reverse order of their registration, except that a function is called after any previously registered functions that had already been called at the time it was registered. If, during the call to any such CX function, a call to the *longjmp*( ) or *siglongjmp*( ) function is made that would terminate the call to the registered function, the behavior is undefined.

If a function registered by a call to *at_quick_exit*( ) fails to return, the remaining registered functions shall not be called and the rest of the *quick_exit*( ) processing shall not be completed.

Finally, the *quick_exit*( ) function shall terminate the process as if by a call to *_Exit*(*status*).

RETURN VALUE

The *quick_exit*( ) function does not return.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

_Exit( ), *at_quick_exit*( ), *atexit*( ), *exit*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

60861 **NAME**

60862 raise — send a signal to the executing process

60863 **SYNOPSIS**

60864 `#include <signal.h>`

60865 `int raise(int sig);`

60866 **DESCRIPTION**

60867 CX The functionality described on this reference page is aligned with the ISO C standard. Any
60868 conflict between the requirements described here and the ISO C standard is unintentional. This
60869 volume of POSIX.1-2024 defers to the ISO C standard.

60870 CX The *raise*( ) function shall send the signal *sig* to the executing thread or process. If a signal
60871 handler is called, the *raise*( ) function shall not return until after the signal handler does.

60872 CX The effect of the *raise*( ) function shall be equivalent to calling:

60873 `pthread_kill(pthread_self(), sig);`

60874 **RETURN VALUE**

60875 CX Upon successful completion, 0 shall be returned. Otherwise, a non-zero value shall be returned
60876 and *errno* shall be set to indicate the error.

60877 **ERRORS**

60878 The *raise*( ) function shall fail if:

60879 CX [EINVAL] The value of the *sig* argument is an invalid signal number.

60880 **EXAMPLES**

60881 None.

60882 **APPLICATION USAGE**

60883 None.

60884 **RATIONALE**

60885 The term ``thread'' is an extension to the ISO C standard.

60886 **FUTURE DIRECTIONS**

60887 None.

60888 **SEE ALSO**

60889 *kill*( ), *sigaction*( )

60890 XBD **<signal.h>**, **<sys/types.h>**

60891 **CHANGE HISTORY**

60892 First released in Issue 4. Derived from the ANSI C standard.

60893 **Issue 5**

60894 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

60895 **Issue 6**

60896 Extensions beyond the ISO C standard are marked.

60897 The following new requirements on POSIX implementations derive from alignment with the
60898 Single UNIX Specification:

60899 • In the RETURN VALUE section, the requirement to set *errno* on error is added.

- The [EINVAL] error condition is added.

Issue 7

Functionality relating to the Threads option is moved to the Base.

60903 **NAME**

60904 rand, srand — pseudo-random number generator

60905 **SYNOPSIS**

60906
```
#include <stdlib.h>
```

60907
```
int rand(void);
```
60908
```
void srand(unsigned seed);
```

60909 **DESCRIPTION**

60910 CX The functionality described on this reference page is aligned with the ISO C standard. Any
60911 conflict between the requirements described here and the ISO C standard is unintentional. This
60912 volume of POSIX.1-2024 defers to the ISO C standard.

60913 The *rand*( ) function shall compute a sequence of pseudo-random integers in the range
60914 XSI [0,{RAND_MAX}] with a period of at least 2^{32}.

60915 The *rand*( ) function need not be thread-safe; however, *rand*( ) shall avoid data races with all
60916 functions other than non-thread-safe pseudo-random sequence generation functions.

60917 The *srand*( ) function uses the argument as a seed for a new sequence of pseudo-random
60918 numbers to be returned by subsequent calls to *rand*( ). If *srand*( ) is then called with the same
60919 seed value, the sequence of pseudo-random numbers shall be repeated. If *rand*( ) is called before
60920 any calls to *srand*( ) are made, the same sequence shall be generated as when *srand*( ) is first
60921 called with a seed value of 1.

60922 The *srand*( ) function need not be thread-safe; however, *srand*( ) shall avoid data races with all
60923 functions other than non-thread-safe pseudo-random sequence generation functions.

60924 The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls
60925 *rand*( ) or *srand*( ).

60926 **RETURN VALUE**

60927 The *rand*( ) function shall return the next pseudo-random number in the sequence.

60928 The *srand*( ) function shall not return a value.

60929 **ERRORS**

60930 No errors are defined.

60931 **EXAMPLES**

60932 **Generating a Pseudo-Random Number Sequence**

60933 The following example demonstrates how to generate a sequence of pseudo-random numbers.

```
60934 #include <stdio.h>
60935 #include <stdlib.h>
60936 ...
60937     long count, i;
60938     char *keystr;
60939     int elementlen, len;
60940     char c;
60941 ...
60942 /* Initial random number generator. */
60943     srand(1);
60944     /* Create keys using only lowercase characters */
60945     len = 0;
60946     for (i=0; i<count; i++) {
```



```
            while (len < elementlen) {
                c = (char) (rand() % 128);
                if (islower(c))
                    keystr[len++] = c;
            }

            keystr[len] = '\0';
            printf("%s Element%0*ld\n", keystr, elementlen, i);
            len = 0;
        }
```

Generating the Same Sequence on Different Machines

The following code defines a pair of functions that could be incorporated into applications wishing to ensure that the same sequence of numbers is generated across different machines.

```
static unsigned long next = 1;
int myrand(void)  /* RAND_MAX assumed to be 32767. */
{
    next = next * 1103515245 + 12345;
    return((unsigned)(next/65536) % 32768);
}

void mysrand(unsigned seed)
{
    next = seed;
}
```

APPLICATION USAGE

These functions should be avoided whenever non-trivial requirements (including safety) have to be fulfilled, unless seeded using *getentropy*( ).

The *drand48*( ) and *random*( ) functions provide much more elaborate pseudo-random number generators.

RATIONALE

The ISO C standard *rand*( ) and *srand*( ) functions allow per-process pseudo-random streams shared by all threads. Those two functions need not change, but there has to be mutual-exclusion that prevents interference between two threads concurrently accessing the random number generator.

With regard to *rand*( ), there are two different behaviors that may be wanted in a multi-threaded program:

1. A single per-process sequence of pseudo-random numbers that is shared by all threads that call *rand*( )
2. A different sequence of pseudo-random numbers for each thread that calls *rand*( )

This is provided by the modified thread-safe function based on whether the seed value is global to the entire process or local to each thread.

This does not address the known deficiencies of the *rand*( ) function implementations, which have been approached by maintaining more state. In effect, this specifies new thread-safe forms of a deficient function.

FUTURE DIRECTIONS

None.

SEE ALSO

drand48( ), *getentropy*( ), *initstate*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The *rand_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *rand*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

Extensions beyond the ISO C standard are marked.

The *rand_r*( ) function is marked as part of the Thread-Safe Functions option.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

The *rand_r*( ) function is marked obsolescent.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0301 [743] is applied.

Issue 8

Austin Group Defect 1134 is applied, adding *getentropy*( ).

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018 standard.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

61011 **NAME**

61012 random — generate pseudo-random number

61013 **SYNOPSIS**

61014 XSI
```
#include <stdlib.h>

long random(void);
```

61016 **DESCRIPTION**

61017 Refer to *initstate( )*.

61018 **NAME**

61019 pread, read — read from a file

61020 **SYNOPSIS**

61021 ```
#include <unistd.h>

ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);
ssize_t read(int fildes, void *buf, size_t nbyte);
```

61024 **DESCRIPTION**

61025 The *read*( ) function shall attempt to read *nbyte* bytes from the file associated with the open file
61026 descriptor, *fildes*, into the buffer pointed to by *buf*. The behavior of multiple concurrent reads on
61027 the same pipe, FIFO, or terminal device is unspecified.

61028 Before any action described below is taken, and if *nbyte* is zero, the *read*( ) function may detect
61029 and return errors as described below. In the absence of errors, or if error detection is not
61030 performed, the *read*( ) function shall return zero and have no other results.

61031 On files that support seeking (for example, a regular file), the *read*( ) shall start at a position in
61032 the file given by the file offset associated with *fildes*. The file offset shall be incremented by the
61033 number of bytes actually read.

61034 Files that do not support seeking—for example, terminals—always read from the current
61035 position. The value of a file offset associated with such a file is undefined.

61036 No data transfer shall occur past the current end-of-file. If the starting position is at or after the
61037 end-of-file, 0 shall be returned. If the file refers to a device special file, the result of subsequent
61038 *read*( ) requests is implementation-defined.

61039 If the value of *nbyte* is greater than {SSIZE_MAX}, the result is implementation-defined.

61040 When attempting to read from an empty pipe or FIFO:

61041 • If no process has the pipe open for writing, *read*( ) shall return 0 to indicate end-of-file.

61042 • If some process has the pipe open for writing and O_NONBLOCK is set, *read*( ) shall return
61043 –1 and set *errno* to [EAGAIN].

61044 • If some process has the pipe open for writing and O_NONBLOCK is clear, *read*( ) shall
61045 block the calling thread until some data is written or the pipe is closed by all processes that
61046 had the pipe open for writing.

61047 When attempting to read a file (other than a pipe or FIFO) that supports non-blocking reads and
61048 has no data currently available:

61049 • If O_NONBLOCK is set, *read*( ) shall return –1 and set *errno* to [EAGAIN].

61050 • If O_NONBLOCK is clear, *read*( ) shall block the calling thread until some data becomes
61051 available.

61052 • The use of the O_NONBLOCK flag has no effect if there is some data available.

61053 The *read*( ) function reads data previously written to a file. If any portion of a regular file prior to
61054 the end-of-file has not been written, *read*( ) shall return bytes with value 0. For example, *lseek*( )
61055 allows the file offset to be set beyond the end of existing data in the file. If data is later written at
61056 this point, subsequent reads in the gap between the previous end of data and the newly written
61057 data shall return bytes with value 0 until data is written into the gap.

61058 Upon successful completion, where *nbyte* is greater than 0, *read*( ) shall mark for update the last
61059 data access timestamp of the file, and shall return the number of bytes read. This number shall
61060 never be greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes

  
```

left in the file is less than *nbyte*, if the *read*( ) request was interrupted by a signal, or if the file is a pipe or FIFO or special file and has fewer than *nbyte* bytes immediately available for reading. For example, a *read*( ) from a file associated with a terminal may return one typed line of data.

If a *read*( ) is interrupted by a signal before it reads any data, it shall return –1 with *errno* set to [EINTR].

If a *read*( ) is interrupted by a signal after it has successfully read some data, it shall return the number of bytes read.

For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *fildes*.

If *fildes* refers to a socket, *read*( ) shall be equivalent to *recv*( ) with no flags set.

SIO If the O_DSYNC and O_RSYNC bits have been set, read I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion. If the O_SYNC and O_RSYNC bits have been set, read I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.

SHM If *fildes* refers to a shared memory object, the result of the *read*( ) function is unspecified.

TYM If *fildes* refers to a typed memory object, the result of the *read*( ) function is unspecified.

The *pread*( ) function shall be equivalent to *read*( ), except that it shall read from a given position in the file without changing the file offset. The first three arguments to *pread*( ) are the same as *read*( ) with the addition of a fourth argument *offset* for the desired position inside the file. An attempt to perform a *pread*( ) on a file that is incapable of seeking shall result in an error.

RETURN VALUE

Upon successful completion, these functions shall return a non-negative integer indicating the number of bytes actually read. Otherwise, the functions shall return –1 and set *errno* to indicate the error.

ERRORS

These functions shall fail if:

[EAGAIN] The file is neither a pipe, nor a FIFO, nor a socket, the O_NONBLOCK flag is set for the file descriptor, and the thread would be delayed in the read operation.

[EBADF] The *fildes* argument is not a valid file descriptor open for reading.

[EINTR] The read operation was terminated due to the receipt of a signal, and no data was transferred.

[EIO] The process is a member of a background process group attempting to read from its controlling terminal, and either the calling thread is blocking SIGTTIN or the process is ignoring SIGTTIN or the process group of the process is orphaned. This error may also be generated for implementation-defined reasons.

XSI [EISDIR] The *fildes* argument refers to a directory and the implementation does not allow the directory to be read using *read*( ) or *pread*( ). The *readdir*( ) function should be used instead.

[EOVERFLOW] The file is a regular file, *nbyte* is greater than 0, the starting position is before the end-of-file, and the starting position is greater than or equal to the offset maximum established in the open file description associated with *fildes*.

The *pread*( ) function shall fail if:

[EINVAL] The file is a regular file or block special file, and the *offset* argument is negative. The file offset shall remain unchanged.

[ESPIPE] The file is incapable of seeking.

The *read*( ) function shall fail if:

[EAGAIN] The file is a pipe or FIFO, the O_NONBLOCK flag is set for the file descriptor, and the thread would be delayed in the read operation.

[EAGAIN] or [EWOULDBLOCK]
The file is a socket, the O_NONBLOCK flag is set for the file descriptor, and the thread would be delayed in the read operation.

[ECONNRESET] A read was attempted on a socket and the connection was forcibly closed by its peer.

[ENOTCONN] A read was attempted on a socket that is not connected.

[ETIMEDOUT] A read was attempted on a socket and a transmission timeout occurred.

These functions may fail if:

[EIO] A physical I/O error has occurred.

[ENOBUFS] Insufficient resources were available in the system to perform the operation.

[ENOMEM] Insufficient memory was available to fulfill the request.

[ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.

EXAMPLES

Reading Data into a Buffer

The following example reads data from the file associated with the file descriptor *fd* into the buffer pointed to by *buf*.

```
#include <sys/types.h>
#include <unistd.h>
...
char buf[20];
size_t nbytes;
ssize_t bytes_read;
int fd;
...
nbytes = sizeof(buf);
bytes_read = read(fd, buf, nbytes);
...
```

APPLICATION USAGE

None.

RATIONALE

This volume of POSIX.1-2024 does not specify the value of the file offset after an error is returned; there are too many cases. For programming errors, such as [EBADF], the concept is meaningless since no file is involved. For errors that are detected immediately, such as [EAGAIN], clearly the offset should not change. After an interrupt or hardware error, however, an updated value would be very useful and is the behavior of many implementations.

Note that a *read*( ) of zero bytes does not modify the last data access timestamp. A *read*( ) that requests more than zero bytes, but returns zero, is required to modify the last data access timestamp.

Implementations are allowed, but not required, to perform error checking for *read*( ) requests of zero bytes.

Input and Output

The use of I/O with large byte counts has always presented problems. Ideas such as *lread*( ) and *lwrite*( ) (using and returning **long**s) were considered at one time. The current solution is to use abstract types on the ISO C standard function to *read*( ) and *write*( ). The abstract types can be declared so that existing functions work, but can also be declared so that larger types can be represented in future implementations. It is presumed that whatever constraints limit the maximum range of **size_t** also limit portable I/O requests to the same range. This volume of POSIX.1-2024 also limits the range further by requiring that the byte count be limited so that a signed return value remains meaningful. Since the return type is also a (signed) abstract type, the byte count can be defined by the implementation to be larger than an **int** can hold.

The standard developers considered adding atomicity requirements to a pipe or FIFO, but recognized that due to the nature of pipes and FIFOs there could be no guarantee of atomicity of reads of {PIPE_BUF} or any other size that would be an aid to applications portability.

This volume of POSIX.1-2024 requires that no action be taken for *read*( ) or *write*( ) when *nbyte* is zero. This is not intended to take precedence over detection of errors (such as invalid buffer pointers or file descriptors). This is consistent with the rest of this volume of POSIX.1-2024, but the phrasing here could be misread to require detection of the zero case before any other errors. A value of zero is to be considered a correct value, for which the semantics are a no-op.

I/O is intended to be atomic to ordinary files and pipes and FIFOs. Atomic means that all the bytes from a single operation that started out together end up together, without interleaving from other I/O operations. It is a known attribute of terminals that this is not honored, and terminals are explicitly (and implicitly permanently) excepted, making the behavior unspecified. The behavior for other device types is also left unspecified, but the wording is intended to imply that future standards might choose to specify atomicity (or not).

There were recommendations to add format parameters to *read*( ) and *write*( ) in order to handle networked transfers among heterogeneous file system and base hardware types. Such a facility may be required for support by the OSI presentation of layer services. However, it was determined that this should correspond with similar C-language facilities, and that is beyond the scope of this volume of POSIX.1-2024. The concept was suggested to the developers of the ISO C standard for their consideration as a possible area for future work.

In 4.3 BSD, a *read*( ) or *write*( ) that is interrupted by a signal before transferring any data does not by default return an [EINTR] error, but is restarted. In 4.2 BSD, 4.3 BSD, and the Eighth Edition, there is an additional function, *select*( ), whose purpose is to pause until specified activity (data to read, space to write, and so on) is detected on specified file descriptors. It is common in applications written for those systems for *select*( ) to be used before *read*( ) in situations (such as keyboard input) where interruption of I/O due to a signal is desired.

The issue of which files or file types are interruptible is considered an implementation design issue. This is often affected primarily by hardware and reliability issues.

There are no references to actions taken following an ``unrecoverable error''. It is considered beyond the scope of this volume of POSIX.1-2024 to describe what happens in the case of hardware errors.

Earlier versions of this standard allowed two very different behaviors with regard to the handling of interrupts. In order to minimize the resulting confusion, it was decided that POSIX.1-2024 should support only one of these behaviors. Historical practice on AT&T-derived systems was to have *read*( ) and *write*( ) return –1 and set *errno* to [EINTR] when interrupted after some, but not all, of the data requested had been transferred. However, the US Department of Commerce FIPS 151-1 and FIPS 151-2 require the historical BSD behavior, in which *read*( ) and *write*( ) return the number of bytes actually transferred before the interrupt. If –1 is returned when any data is transferred, it is difficult to recover from the error on a seekable device and impossible on a non-seekable device. Most new implementations support this behavior. The behavior required by POSIX.1-2024 is to return the number of bytes transferred.

POSIX.1-2024 does not specify when an implementation that buffers *read*( )s actually moves the data into the user-supplied buffer, so an implementation may choose to do this at the latest possible moment. Therefore, an interrupt arriving earlier may not cause *read*( ) to return a partial byte count, but rather to return –1 and set *errno* to [EINTR].

Consideration was also given to combining the two previous options, and setting *errno* to [EINTR] while returning a short count. However, not only is there no existing practice that implements this, it is also contradictory to the idea that when *errno* is set, the function responsible shall return –1.

This volume of POSIX.1-2024 intentionally does not specify any *pread*( ) errors related to pipes, FIFOs, and sockets other than [ESPIPE].

FUTURE DIRECTIONS

None.

SEE ALSO

fcntl( ), *lseek*( ), *open*( ), *pipe*( ), *readv*( )

XBD Chapter 11 (on page 199), **<sys/uio.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Large File Summit extensions are added.

The *pread*( ) function is added.

Issue 6

The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are marked as part of the XSI STREAMS Option Group.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION now states that if *read*( ) is interrupted by a signal after it has successfully read some data, it returns the number of bytes read. In Issue 3, it was optional whether *read*( ) returned the number of bytes read, or whether it returned –1 with *errno* set to [EINTR]. This is a FIPS requirement.
- In the DESCRIPTION, text is added to indicate that for regular files, no data transfer occurs past the offset maximum established in the open file description associated with *fildes*. This change is to support large files.

- The [EOVERFLOW] mandatory error condition is added.
- The [ENXIO] optional error condition is added.

Text referring to sockets is added to the DESCRIPTION.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effect of reading zero bytes is clarified.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *read*( ) results are unspecified for typed memory objects.

New RATIONALE is added to explain the atomicity requirements for input and output operations.

The following error conditions are added for operations on sockets: [EAGAIN], [ECONNRESET], [ENOTCONN], and [ETIMEDOUT].

The [EIO] error is made optional.

The following error conditions are added for operations on sockets: [ENOBUFS] and [ENOMEM].

The *readv*( ) function is split out into a separate reference page.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/108 is applied, updating the [EAGAIN] error in the ERRORS section from ``the process would be delayed'' to ``the thread would be delayed''.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/109 is applied, making an editorial correction in the RATIONALE section.

Issue 7

The *pread*( ) function is moved from the XSI option to the Base.

Functionality relating to the XSI STREAMS option is marked obsolescent.

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0480 [218], XSH/TC1-2008/0481 [79], XSH/TC1-2008/0482 [218], XSH/TC1-2008/0483 [218], XSH/TC1-2008/0484 [218], and XSH/TC1-2008/0485 [218,428] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0302 [710] and XSH/TC2-2008/0303 [676,710] are applied.

Issue 8

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

61268 **NAME**

61269 readddir, readdir_r — read a directory

61270 **SYNOPSIS**

61271 ```
#include <dirent.h>
```

61272 ```
struct dirent *readdir(DIR *dirp);
```

61273 OB ```
int readdir_r(DIR *restrict dirp, struct dirent *restrict entry,
```

61274 ```
    struct dirent **restrict result);
```

61275 **DESCRIPTION**

61276 The type **DIR**, which is defined in the **<dirent.h>** header, represents a *directory stream*, which is
61277 an ordered sequence of all the directory entries in a particular directory. Directory entries
61278 represent files; files may be removed from a directory or added to a directory asynchronously to
61279 the operation of *readdir*( ).

61280 The *readdir*( ) function shall return a pointer to a structure representing the directory entry at the
61281 current position in the directory stream specified by the argument *dirp*, and position the
61282 directory stream at the next entry. It shall return a null pointer upon reaching the end of the
61283 directory stream. The structure **dirent** defined in the **<dirent.h>** header describes a directory
61284 entry. The value of the structure's *d_ino* member shall be set to the file serial number of the file
61285 named by the *d_name* member. If the *d_name* member names a symbolic link, the value of the
61286 *d_ino* member shall be set to the file serial number of the symbolic link itself. The *d_name*
61287 member shall be a filename string, and (if not dot or dot-dot) shall contain the same byte
61288 sequence as the last pathname component of the string used to create the directory entry, plus
61289 the terminating <NUL> byte.

61290 The *readdir*( ) function shall not return directory entries containing empty names. If entries for
61291 dot or dot-dot exist, one entry shall be returned for dot and one entry shall be returned for dot-
61292 dot; otherwise, they shall not be returned.

61293 The application shall not modify the structure to which the return value of *readdir*( ) points, nor
61294 any storage areas pointed to by pointers within the structure. The returned pointer, and pointers
61295 within the structure, might be invalidated or the structure or the storage areas might be
61296 overwritten by a subsequent call to *readdir*( ) on the same directory stream. They shall not be
61297 affected by a call to *readdir*( ) on a different directory stream. The returned pointer, and pointers
61298 within the structure, might also be invalidated if the calling thread is terminated.

61299 If a file is removed from or added to the directory after the most recent call to *opendir*( ) or
61300 *rewinddir*( ), whether a subsequent call to *readdir*( ) returns an entry for that file is unspecified.

61301 The *readdir*( ) function may buffer several directory entries per actual read operation; *readdir*( )
61302 shall mark for update the last data access timestamp of the directory each time the directory is
61303 actually read.

61304 After a call to *fork*( ), either the parent or child (but not both) may continue processing the
61305 XSI directory stream using *readdir*( ), *rewinddir*( ), or *seekdir*( ). If both the parent and child processes
61306 use these functions, the result is undefined.

61307 The *readdir*( ) function need not be thread-safe if concurrent calls are made for the same directory
61308 stream.

61309 Applications wishing to check for error situations should set *errno* to 0 before calling *readdir*( ). If
61310 *errno* is set to non-zero on return, an error occurred.

61311 OB The *readdir_r*( ) function shall initialize the **dirent** structure referenced by *entry* to represent the
61312 directory entry at the current position in the directory stream referred to by *dirp*, store a pointer

to this structure at the location referenced by *result*, and position the directory stream at the next entry.

The storage pointed to by *entry* shall be large enough for a **dirent** with an array of **char** *d_name* members containing at least {NAME_MAX}+1 elements.

Upon successful return, the pointer returned at **result* shall have the same value as the argument *entry*. Upon reaching the end of the directory stream, this pointer shall have the value NULL.

The *readdir_r*( ) function shall not return directory entries containing empty names.

If a file is removed from or added to the directory after the most recent call to *opendir*( ) or *rewinddir*( ), whether a subsequent call to *readdir_r*( ) returns an entry for that file is unspecified.

The *readdir_r*( ) function may buffer several directory entries per actual read operation; *readdir_r*( ) shall mark for update the last data access timestamp of the directory each time the directory is actually read.

RETURN VALUE

Upon successful completion, *readdir*( ) shall return a pointer to an object of type **struct dirent**. When an error is encountered, a null pointer shall be returned and *errno* shall be set to indicate the error. When the end of the directory is encountered, a null pointer shall be returned and *errno* is not changed.

OB If successful, the *readdir_r*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

OB The *readdir*( ) and *readdir_r*( ) functions shall fail if:

[EOVERFLOW] One of the values in the structure to be returned cannot be represented correctly.

[ENOMEM] Insufficient memory is available.

OB The *readdir*( ) and *readdir_r*( ) functions may fail if:

[EBADF] The *dirp* argument does not refer to an open directory stream.

[ENOENT] The current position of the directory stream is invalid.

EXAMPLES

The following sample program searches the current directory for each of the arguments supplied on the command line.

```
#include <dirent.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>

static void lookup(const char *arg)
{
    DIR *dirp;
    struct dirent *dp;

    if ((dirp = opendir(".")) == NULL) {
        perror("couldn't open '.'");
        return;
    }

    do {
```



```
61356                errno = 0;
61357                if ((dp = readdir(dirp)) != NULL) {
61358                    if (strcmp(dp->d_name, arg) != 0)
61359                        continue;
61360                    (void) printf("found %s\n", arg);
61361                    (void) closedir(dirp);
61362                        return;
61363                }
61364            } while (dp != NULL);
61365            if (errno != 0)
61366                perror("error reading directory");
61367            else
61368                (void) printf("failed to find %s\n", arg);
61369            (void) closedir(dirp);
61370            return;
61371        }
61372        int main(int argc, char *argv[])
61373        {
61374            int i;
61375            for (i = 1; i < argc; i++)
61376                lookup(argv[i]);
61377            return (0);
61378        }
```

61379 **APPLICATION USAGE**

61380 The *readdir*( ) function should be used in conjunction with *opendir*( ), *closedir*( ), and *rewinddir*( ) to
61381 examine the contents of the directory.

61382 The *readdir_r*( ) function returns values in a user-supplied buffer, but does not allow the size of
61383 the buffer to be specified by the caller. If {NAME_MAX} is indeterminate, there is no way for an
61384 application to know how large the buffer needs to be and *readdir_r*( ) cannot safely be used.

61385 **RATIONALE**

61386 The returned value of *readdir*( ) merely *represents* a directory entry. No equivalence should be
61387 inferred.

61388 Historical implementations of *readdir*( ) obtain multiple directory entries on a single read
61389 operation, which permits subsequent *readdir*( ) operations to operate from the buffered
61390 information. Any wording that required each successful *readdir*( ) operation to mark the
61391 directory last data access timestamp for update would disallow such historical performance-
61392 oriented implementations.

61393 When returning a directory entry for the root of a mounted file system, some historical
61394 implementations of *readdir*( ) returned the file serial number of the underlying mount point,
61395 rather than of the root of the mounted file system. This behavior is considered to be a bug, since
61396 the underlying file serial number has no significance to applications.

61397 Since *readdir*( ) returns NULL when it detects an error and when the end of the directory is
61398 encountered, an application that needs to tell the difference must set *errno* to zero before the call
61399 and check it if NULL is returned. Since the function must not change *errno* in the second case
61400 and must set it to a non-zero value in the first case, a zero *errno* after a call returning NULL
61401 indicates end-of-directory; otherwise, an error.

Routines to deal with this problem more directly were proposed:

```
int derror (dirp)
DIR *dirp;

void clearderr (dirp)
DIR *dirp;
```

The first would indicate whether an error had occurred, and the second would clear the error indication. The simpler method involving *errno* was adopted instead by requiring that *readdir*( ) not change *errno* when end-of-directory is encountered.

An error or signal indicating that a directory has changed while open was considered but rejected.

Historically, *readdir*( ) returned a pointer to an internal static buffer that was overwritten by each call. The *readdir_r*( ) function was added as a thread-safe alternative that returns values in a user-supplied buffer. However, it does not allow the size of the buffer to be specified by the caller, and so is only usable if {NAME_MAX} is a compile-time constant or *fpathconf*( ) with _SC_NAME_MAX returns a value other than –1. If {NAME_MAX} is indeterminate (indicated by *fpathconf*( ) returning –1), there is no way to reliably allocate a buffer large enough to hold a filename being returned by *readdir_r*( ). Therefore, *readdir_r*( ) has been marked obsolescent and *readdir*( ) is now required to be thread safe as long as there are no concurrent calls to it on a single directory stream.

Conforming file systems are required to store filenames unaltered from how they were created (via *open*( ), *link*( ), *mkdir*( ), *mkfifo*( ), *rename*( ), etc.). By definition, a filename string does not include a <slash>, even if a trailing <slash> was present in the pathname presented to *mkdir*( ) when creating a sub-directory.

However, there are non-conforming file systems where filenames are converted to a canonical representation before a directory entry is created, such that it is possible to create a file using one string, then perform *opendir*( ) and a *readdir*( ) loop and not encounter the same string, because *readdir*( ) returns the canonical form of the string instead. Such non-conforming file systems also have the issue that multiple filenames can resolve to the same directory entry, with potentially confusing results. This standard cannot mandate the behavior of non-conforming file systems, and strictly conforming applications need not worry about dealing with such file systems, but it is a concern for developers of portable applications. Therefore, this standard recommends that file system implementations that perform canonicalization of filenames should reject attempts to create a directory entry with a non-canonical filename using the [EILSEQ] error. However, if a directory entry already exists, it is reasonable for a file system to permit accessing that file via a non-canonical filename.

FUTURE DIRECTIONS

The *readdir_r*( ) function may be removed in a future version.

SEE ALSO

closedir( ), *dirfd*( ), *exec*, *fdopendir*( ), *fstatat*( ), *posix_getdents*( ), *rewinddir*( ), *symlink*( )

XBD **<dirent.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 2.

Issue 5

Large File Summit extensions are added.

The *readdir_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *readdir*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *readdir_r*( ) function is marked as part of the Thread-Safe Functions option.

The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir_r*( ).

The Open Group Corrigendum U026/8 is applied, clarifying the wording of the successful return for the *readdir_r*( ) function.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- A statement is added to the DESCRIPTION indicating the disposition of certain fields in **struct dirent** when an entry refers to a symbolic link.
- The [EOVERFLOW] mandatory error condition is added. This change is to support large files.
- The [ENOENT] optional error condition is added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

The **restrict** keyword is added to the *readdir_r*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/50 is applied, replacing the EXAMPLES section with a new example.

Issue 7

Austin Group Interpretation 1003.1-2001 #059 is applied, updating the ERRORS section.

Austin Group Interpretation 1003.1-2001 #156 is applied.

The *readdir_r*( ) function is moved from the Thread-Safe Functions option to the Base.

Changes are made related to support for finegrained timestamps.

The value of the *d_ino* member is no longer unspecified for symbolic links.

SD5-XSH-ERN-193 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0486 [75] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0304 [656] is applied.

Issue 8

Austin Group Defect 293 is applied, adding a requirement that *d_name* contains the same byte sequence as the last pathname component of the string used to create the directory entry.

Austin Group Defects 696 and 1664 are applied, making *readdir_r*( ) obsolescent, requiring *readdir*( ) to be thread-safe except when concurrent calls are made for the same directory stream,

and adding the [ENOMEM] error.

Austin Group Defect 697 is applied, adding *posix_getdents*( ) to the SEE ALSO section.

61486 **NAME**

61487 readlink, readlinkat — read the contents of a symbolic link

61488 **SYNOPSIS**

61489
```
#include <unistd.h>
```

61490
```
ssize_t readlink(const char *restrict path, char *restrict buf,
```
61491
```
    size_t bufsize);
```

61492 OH
```
#include <fcntl.h>
```

61493
```
ssize_t readlinkat(int fd, const char *restrict path,
```
61494
```
    char *restrict buf, size_t bufsize);
```

61495 **DESCRIPTION**

61496 The *readlink*( ) function shall place the contents of the symbolic link referred to by *path* in the
61497 buffer *buf* which has size *bufsize*. If the number of bytes in the symbolic link is less than *bufsize*,
61498 the contents of the remainder of *buf* are unspecified. If the *buf* argument is not large enough to
61499 contain the link content, the first *bufsize* bytes shall be placed in *buf*.

61500 If the value of *bufsize* is greater than {SSIZE_MAX}, the result is implementation-defined.

61501 Upon successful completion, *readlink*( ) shall mark for update the last data access timestamp of
61502 the symbolic link.

61503 The *readlinkat*( ) function shall be equivalent to the *readlink*( ) function except in the case where
61504 *path* specifies a relative path. In this case the symbolic link whose content is read is relative to the
61505 directory associated with the file descriptor *fd* instead of the current working directory. If the
61506 access mode of the open file description associated with the file descriptor is not O_SEARCH,
61507 the function shall check whether directory searches are permitted using the current permissions
61508 of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function
61509 shall not perform the check.

61510 If *readlinkat*( ) is passed the special value AT_FDCWD in the *fd* parameter, the current working
61511 directory shall be used and the behavior shall be identical to a call to *readlink*( ).

61512 **RETURN VALUE**

61513 Upon successful completion, these functions shall return the count of bytes placed in the buffer.
61514 Otherwise, these functions shall return a value of –1, leave the buffer unchanged, and set *errno* to
61515 indicate the error.

61516 **ERRORS**

61517 These functions shall fail if:

61518 [EACCES] Search permission is denied for a component of the path prefix of *path*.

61519 [EINVAL] The *path* argument names a file that is not a symbolic link.

61520 [EIO] An I/O error occurred while reading from the file system.

61521 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
61522 argument.

61523 [ENAMETOOLONG]
61524 The length of a component of a pathname is longer than {NAME_MAX}.

61525 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

61526 [ENOTDIR] A component of the path prefix names an existing file that is neither a
61527 directory nor a symbolic link to a directory, or the *path* argument contains at
61528 least one non-<slash> character and ends with one or more trailing <slash>

characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

The *readlinkat*( ) function shall fail if:

[EACCES] The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

[EBADF] The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

[ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

These functions may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

EXAMPLES

Reading the Name of a Symbolic Link

The following example shows how to read the name of a symbolic link named **/modules/pass1**.

```
#include <unistd.h>

char buf[1024];
ssize_t len;
...
if ((len = readlink("/modules/pass1", buf, sizeof(buf)-1)) != -1)
    buf[len] = '\0';
```

APPLICATION USAGE

Conforming applications should not assume that the returned contents of the symbolic link are null-terminated.

RATIONALE

The type associated with *bufsiz* is a **size_t** in order to be consistent with both the ISO C standard and the definition of *read*( ). The behavior specified for *readlink*( ) when *bufsiz* is zero represents historical practice. For this case, the standard developers considered a change whereby *readlink*( ) would return the number of non-null bytes contained in the symbolic link with the buffer *buf* remaining unchanged; however, since the **stat** structure member *st_size* value can be used to determine the size of buffer necessary to contain the contents of the symbolic link as returned by *readlink*( ), this proposal was rejected, and the historical practice retained.

The purpose of the *readlinkat*( ) function is to read the content of symbolic links in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *readlink*( ), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *readlinkat*( ) function it can be guaranteed that the symbolic link read is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

fstatat( ), *symlink( )*

XBD **<fcntl.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The return type is changed to **ssize_t**, to align with the IEEE P1003.1a draft standard.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- This function is made mandatory.
- In this function it is possible for the return value to exceed the range of the type **ssize_t** (since **size_t** has a larger range of positive values than **ssize_t**). A sentence restricting the size of the **size_t** object is added to the description to resolve this conflict.

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- The FUTURE DIRECTIONS section is changed to None.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

The **restrict** keyword is added to the *readlink*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

SD5-XSH-ERN-189 is applied, updating the ERRORS section.

The *readlinkat*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The [EACCES] error is removed from the ``may fail'' error conditions.

Changes are made to allow a directory to be opened for searching.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0487 [120], XSH/TC1-2008/0488 [461], XSH/TC1-2008/0489 [143], XSH/TC1-2008/0490 [324], XSH/TC1-2008/0491 [278], XSH/TC1-2008/0492 [278], XSH/TC1-2008/0493 [455], and XSH/TC1-2008/0494 [151,231] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0305 [591], XSH/TC2-2008/0306 [817], XSH/TC2-2008/0307 [817], and XSH/TC2-2008/0308 [591] are applied.

NAME

readv — read a vector

SYNOPSIS

XSI
```
#include <sys/uio.h>

ssize_t readv(int fildes, const struct iovec *iov, int iovcnt);
```

DESCRIPTION

The *readv*( ) function shall be equivalent to *read*( ), except as described below. The *readv*( ) function shall place the input data into the *iovcnt* buffers specified by the members of the *iov* array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*–1]. The *iovcnt* argument is valid if greater than 0 and less than or equal to {IOV_MAX}.

Each *iovec* entry specifies the base address and length of an area in memory where data should be placed. The *readv*( ) function shall always fill an area completely before proceeding to the next.

Upon successful completion, *readv*( ) shall mark for update the last data access timestamp of the file.

RETURN VALUE

Refer to *read*( ).

ERRORS

Refer to *read*( ).

In addition, the *readv*( ) function shall fail if:

[EINVAL] The sum of the *iov_len* values in the *iov* array overflowed an **ssize_t**.

The *readv*( ) function may fail if:

[EINVAL] The *iovcnt* argument was less than or equal to 0, or greater than {IOV_MAX}.

EXAMPLES

Reading Data into an Array

The following example reads data from the file associated with the file descriptor *fd* into the buffers specified by members of the *iov* array.

```
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
...
ssize_t bytes_read;
int fd;
char buf0[20];
char buf1[30];
char buf2[40];
int iovcnt;
struct iovec iov[3];

iov[0].iov_base = buf0;
iov[0].iov_len = sizeof(buf0);
iov[1].iov_base = buf1;
iov[1].iov_len = sizeof(buf1);
iov[2].iov_base = buf2;
```



```
        iov[2].iov_len = sizeof(buf2);
        ...
        iovcnt = sizeof(iov) / sizeof(struct iovec);
        bytes_read = readv(fd, iov, iovcnt);
        ...
```

APPLICATION USAGE

None.

RATIONALE

Refer to *read( )*.

FUTURE DIRECTIONS

None.

SEE ALSO

read( ), *writev( )*

XBD **<sys/uio.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 6

Split out from the *read( )* reference page.

Issue 7

Changes are made related to support for finegrained timestamps.

NAME

realloc, reallocarray — memory reallocators

SYNOPSIS

```
#include <stdlib.h>

void *realloc(void *ptr, size_t size);
CX void *reallocarray(void *ptr, size_t nelem, size_t elsize);
```

DESCRIPTION

CX For *realloc*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *realloc*() function shall deallocate the old object pointed to by *ptr* and return a pointer to a new object that has the size specified by *size*. The contents of the new object shall be the same as that of the old object prior to deallocation, up to the lesser of the new and old sizes. Any bytes in the new object beyond the size of the old object have indeterminate values.

CX The *reallocarray*() function shall be equivalent to the call `realloc(ptr, nelem * elsize)` except that overflow in the multiplication shall be an error.

CX If *ptr* is a null pointer, *realloc*() or *reallocarray*() shall be equivalent to *malloc*() for the specified size. Otherwise, if *ptr* does not match a pointer earlier returned by *aligned_alloc*(), *calloc*(), ADV *malloc*(), *posix_memalign*(), *realloc*(), CX *reallocarray*(), or a function in POSIX.1-2024 that allocates memory as if by *malloc*(), or if the CX space has been deallocated by a call to *free*(), *reallocarray*(), or *realloc*(), the behavior is undefined.

If *size* is non-zero and memory for the new object is not allocated, the old object shall not be deallocated.

CX The order and contiguity of storage allocated by successive calls to *realloc*() or *reallocarray*() is unspecified. The pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to a pointer to any type of object with a fundamental alignment requirement and then used to access such an object in the space allocated (until the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object disjoint from any other object. The pointer returned shall point to the start (lowest byte address) of the allocated space. If the space cannot be allocated, a null pointer shall be returned.

CX For purposes of determining the existence of a data race, *realloc*() and *reallocarray*() shall each behave as though it accessed only memory locations accessible through its argument and not other static duration storage. The function may, however, visibly modify the storage that it ADV allocates. Calls to *aligned_alloc*(), *calloc*(), *free*(), *malloc*(), *posix_memalign*(), CX *reallocarray*(), and *realloc*() that allocate or deallocate a particular region of memory shall occur in a single total order (see Section 4.15.1, on page 100), and each such deallocation call shall synchronize with the next allocation (if any) in this order.

RETURN VALUE

CX Upon successful completion, *realloc*() and *reallocarray*() shall return a pointer to the new object (which can have the same value as a pointer to the old object), or a null pointer if the new object has not been allocated.

OB If *size* is 0,
OB CX or either *nelem* or *elsize* is 0,
OB either:

61717 OB • A null pointer shall be returned
61718 OB CX and, if *ptr* is not a null pointer, *errno* shall be set to [EINVAL].

61719 OB • A pointer to the allocated space shall be returned, and the memory object pointed to by *ptr*
61720 shall be freed. The application shall ensure that the pointer is not used to access an object.

61721 CX If there is not enough available memory, *realloc*( ) and *reallocarray*( ) shall return a null pointer
61722 CX and set *errno* to [ENOMEM].

61723 **ERRORS**

61724 CX The *realloc*( ) and *reallocarray*( ) functions shall fail if:

61725 CX [ENOMEM] Insufficient memory is available.

61726 CX The *reallocarray*( ) function shall fail if:

61727 [ENOMEM] The calculation *nelem* * *elsize* would overflow.

61728 CX The *realloc*( ) and *reallocarray*( ) functions may fail if:

61729 CX [EINVAL] The requested allocation size is 0 and the implementation does not support 0
61730 sized allocations.

61731 **EXAMPLES**

61732 None.

61733 **APPLICATION USAGE**

61734 The ISO C standard makes it implementation-defined whether a call to *realloc*(*p*, 0) frees the
61735 space pointed to by *p* if it returns a null pointer because memory for the new object was not
61736 allocated. POSIX.1 instead requires that implementations set *errno* if a null pointer is returned
61737 and the space has not been freed, and POSIX applications should only free the space if *errno* was
61738 changed.

61739 **RATIONALE**

61740 See the RATIONALE for *malloc*( ).

61741 **FUTURE DIRECTIONS**

61742 The ISO C standard states that invoking *realloc*( ) with a *size* argument equal to zero is an
61743 obsolescent feature. This feature may be removed in a future version of this standard.

61744 **SEE ALSO**

61745 *aligned_alloc*( ), *calloc*( ), *free*( ), *malloc*( )

61746 XBD **<stdlib.h>**

61747 **CHANGE HISTORY**

61748 First released in Issue 1. Derived from Issue 1 of the SVID.

61749 **Issue 6**

61750 Extensions beyond the ISO C standard are marked.

61751 The following new requirements on POSIX implementations derive from alignment with the
61752 Single UNIX Specification:

61753 • In the RETURN VALUE section, if there is not enough available memory, the setting of
61754 *errno* to [ENOMEM] is added.

61755 • The [ENOMEM] error condition is added.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0495 [400], XSH/TC1-2008/0496 [400], XSH/TC1-2008/0497 [400], and XSH/TC1-2008/0498 [400] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0309 [526] and XSH/TC2-2008/0310 [526,688] are applied.

Issue 8

Austin Group Defect 374 is applied, adding the [EINVAL] error.

Austin Group Defect 1218 is applied, adding *reallocarray*( ).

Austin Group Defect 1302 is applied, aligning the *realloc*( ) function with the ISO/IEC 9899: 2018 standard.

Austin Group Defect 1387 is applied, changing the RATIONALE section.

61767 **NAME**

61768 realpath — resolve a pathname

61769 **SYNOPSIS**

61770 ```
#include <stdlib.h>
```

61771 ```
char *realpath(const char *restrict file_name,
```

61772 ```
    char *restrict resolved_name);
```

61773 **DESCRIPTION**

61774 The *realpath*( ) function shall derive, from the pathname pointed to by *file_name*, an absolute
61775 pathname that resolves to the same directory entry, whose resolution does not involve `'.'`,
61776 `'..'`, or symbolic links. If *resolved_name* is a null pointer, the generated pathname shall be
61777 stored as a null-terminated string in a buffer allocated as if by a call to *malloc*( ). Otherwise, if
61778 {PATH_MAX} is defined as a constant in the **<limits.h>** header, then the generated pathname
61779 shall be stored as a null-terminated string, up to a maximum of {PATH_MAX} bytes, in the
61780 buffer pointed to by *resolved_name*.

61781 If *resolved_name* is not a null pointer and {PATH_MAX} is not defined as a constant in the
61782 **<limits.h>** header, the behavior is undefined.

61783 **RETURN VALUE**

61784 Upon successful completion, *realpath*( ) shall return a pointer to the buffer containing the
61785 resolved name. Otherwise, *realpath*( ) shall return a null pointer and set *errno* to indicate the
61786 error.

61787 If the *resolved_name* argument is a null pointer, the pointer returned by *realpath*( ) can be passed
61788 to *free*( ).

61789 If the *resolved_name* argument is not a null pointer and the *realpath*( ) function fails, the contents
61790 of the buffer pointed to by *resolved_name* are undefined.

61791 **ERRORS**

61792 The *realpath*( ) function shall fail if:

61793 [EACCES] Search permission was denied for a component of the path prefix of *file_name*.

61794 [EINVAL] The *file_name* argument is a null pointer.

61795 [EIO] An error occurred while reading from the file system.

61796 [ELOOP] A loop exists in symbolic links encountered during resolution of the *file_name*
61797 argument.

61798 [ENAMETOOLONG]

61799 The length of a component of a pathname is longer than {NAME_MAX}.

61800 [ENOENT] A component of *file_name* does not name an existing file or *file_name* points to
61801 an empty string.

61802 [ENOTDIR] A component of the path prefix names an existing file that is neither a
61803 directory nor a symbolic link to a directory, or the *file_name* argument contains
61804 at least one non-<slash> character and ends with one or more trailing <slash>
61805 characters and the last pathname component names an existing file that is
61806 neither a directory nor a symbolic link to a directory.

  
```

The *realpath*( ) function may fail if:

[EACCES] The *file_name* argument does not begin with a <slash> and none of the symbolic links (if any) processed during pathname resolution of *file_name* had contents that began with a <slash>, and either search permission was denied for the current directory or read or search permission was denied for a directory above the current directory in the file hierarchy.

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *file_name* argument.

[ENAMETOOLONG] The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

[ENOMEM] Insufficient storage space is available.

EXAMPLES

Generating an Absolute Pathname

The following example generates an absolute pathname for the file identified by the *symlinkpath* argument. The generated pathname is stored in the buffer pointed to by *actualpath*.

```
#include <stdlib.h>
...
char *symlinkpath = "/tmp/symlink/file";
char *actualpath;

actualpath = realpath(symlinkpath, NULL);
if (actualpath != NULL)
{
    ... use actualpath ...

    free(actualpath);
}
else
{
    ... handle error ...
}
```

APPLICATION USAGE

For functions that allocate memory as if by *malloc*( ), the application should release such memory when it is no longer required by a call to *free*( ). For *realpath*( ), this is the return value.

RATIONALE

Since *realpath*( ) has no *length* argument, if {PATH_MAX} is not defined as a constant in **<limits.h>**, applications have no way of determining how large a buffer they need to allocate for it to be safe to pass to *realpath*( ). A {PATH_MAX} value obtained from a prior *pathconf*( ) call is out-of-date by the time *realpath*( ) is called. Hence the only reliable way to use *realpath*( ) when {PATH_MAX} is not defined in **<limits.h>** is to pass a null pointer for *resolved_name* so that *realpath*( ) will allocate a buffer of the necessary size.

FUTURE DIRECTIONS

None.

SEE ALSO

fpathconf( ), *free*( ), *getcwd*( ), *sysconf*( )

XBD **<limits.h>**, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The **restrict** keyword is added to the *realpath*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/51 is applied, adding new text to the DESCRIPTION for the case when *resolved_name* is a null pointer, changing the [EINVAL] error text, adding text to the RATIONALE, and adding text to FUTURE DIRECTIONS.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/110 is applied, updating the ERRORS section to refer to the *file_name* argument, rather than a nonexistent *path* argument.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

This function is updated for passing a null pointer to *realpath*( ) for the *resolved_name* argument.

The APPLICATION USAGE section is updated to clarify that memory is allocated as if by *malloc*( ).

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0499 [353], XSH/TC1-2008/0500 [324], and XSH/TC1-2008/0501 [353] are applied.

Issue 8

Austin Group Defect 1663 is applied, removing XSI shading from *realpath*( ).

61876 **NAME**

61877 recv — receive a message from a connected socket

61878 **SYNOPSIS**

61879 `#include <sys/socket.h>`

61880 `ssize_t recv(int socket, void *buffer, size_t length, int flags);`

61881 **DESCRIPTION**

61882 The *recv*( ) function shall receive a message from a connection-mode or connectionless-mode
61883 socket. It is normally used with connected sockets because it does not permit the application to
61884 retrieve the source address of received data.

61885 The *recv*( ) function takes the following arguments:

61886 *socket* Specifies the socket file descriptor.

61887 *buffer* Points to a buffer where the message should be stored.

61888 *length* Specifies the length in bytes of the buffer pointed to by the *buffer* argument.

61889 *flags* Specifies the type of message reception. Values of this argument are formed by
61890 logically OR'ing zero or more of the following values:

61891 MSG_PEEK Peeks at an incoming message. The data is treated as unread and
61892 the next *recv*( ) or similar function shall still return this data.

61893 MSG_OOB Requests out-of-band data. The significance and semantics of
61894 out-of-band data are protocol-specific.

61895 MSG_WAITALL On SOCK_STREAM sockets this requests that the function block
61896 until the full amount of data can be returned. The function may
61897 return the smaller amount of data if the socket is a message-
61898 based socket, if a signal is caught, if the connection is terminated,
61899 if MSG_PEEK was specified, or if an error is pending for the
61900 socket.

61901 The *recv*( ) function shall return the length of the message written to the buffer pointed to by the
61902 *buffer* argument. For message-based sockets, such as SOCK_DGRAM and SOCK_SEQPACKET,
61903 the entire message shall be read in a single operation. If a message is too long to fit in the
61904 supplied buffer, and MSG_PEEK is not set in the *flags* argument, the excess bytes shall be
61905 discarded. For stream-based sockets, such as SOCK_STREAM, message boundaries shall be
61906 ignored. In this case, data shall be returned to the user as soon as it becomes available, and no
61907 data shall be discarded.

61908 If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first
61909 message.

61910 If no messages are available at the socket and O_NONBLOCK is not set on the socket's file
61911 descriptor, *recv*( ) shall block until a message arrives or a timeout occurs (see SO_RCVTIMEO in
61912 Section 2.10.16, on page 554). If no messages are available at the socket and O_NONBLOCK is
61913 set on the socket's file descriptor, *recv*( ) shall fail and set *errno* to [EAGAIN] or
61914 [EWOULDBLOCK].

61915 **RETURN VALUE**

61916 Upon successful completion, *recv*( ) shall return the length of the message in bytes. If no
61917 messages are available to be received and the peer has performed an orderly shutdown, *recv*( )
61918 shall return 0. Otherwise, –1 shall be returned and *errno* set to indicate the error.

61919 **ERRORS**

61920 The *recv*( ) function shall fail if:

61921 [EAGAIN] or [EWOULDBLOCK]

61922 The socket's file descriptor is marked O_NONBLOCK and no data is waiting
61923 to be received; or MSG_OOB is set and no out-of-band data is available and
61924 either the socket's file descriptor is marked O_NONBLOCK or the socket does
61925 not support blocking to await out-of-band data. See also SO_RCVTIMEO in
61926 Section 2.10.16 (on page 554).

61927 [EBADF] The *socket* argument is not a valid file descriptor.

61928 [ECONNRESET] A connection was forcibly closed by a peer.

61929 [EINTR] The *recv*( ) function was interrupted by a signal that was caught, before any
61930 data was available.

61931 [EINVAL] The MSG_OOB flag is set and no out-of-band data is available.

61932 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

61933 [ENOTSOCK] The *socket* argument does not refer to a socket.

61934 [EOPNOTSUPP] The specified flags are not supported for this socket type or protocol.

61935 [ETIMEDOUT] The connection timed out during connection establishment, or due to a
61936 transmission timeout on active connection.

61937 The *recv*( ) function may fail if:

61938 [EIO] An I/O error occurred while reading from or writing to the file system.

61939 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

61940 [ENOMEM] Insufficient memory was available to fulfill the request.

61941 **EXAMPLES**

61942 None.

61943 **APPLICATION USAGE**

61944 The *recv*( ) function is equivalent to *recvfrom*( ) with null pointer *address* and *address_len*
61945 arguments, and to *read*( ) if the *socket* argument refers to a socket and the *flags* argument is 0.

61946 The *select*( ) and *poll*( ) functions can be used to determine when data is available to be received.

61947 **RATIONALE**

61948 None.

61949 **FUTURE DIRECTIONS**

61950 None.

61951 **SEE ALSO**

61952 *poll*( ), *pselect*( ), *read*( ), *recvmsg*( ), *recvfrom*( ), *send*( ), *sendmsg*( ), *sendto*( ), *setsockopt*( ), *shutdown*( ),
61953 *socket*( ), *write*( )

61954 XBD **<sys/socket.h>**

61955 **CHANGE HISTORY**

61956 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

61957 **Issue 7**

61958 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0502 [462] is applied.

61959 **Issue 8**

61960 Austin Group Defect 1429 is applied, clarifying the behavior on timeout by adding references to
61961 Section 2.10.16 (on page 554).

61962 **NAME**

61963 recvfrom — receive a message from a socket

61964 **SYNOPSIS**

61965
```
#include <sys/socket.h>
```

61966
```
ssize_t recvfrom(int socket, void *restrict buffer, size_t length,
    int flags, struct sockaddr *restrict address,
    socklen_t *restrict address_len);
```

61969 **DESCRIPTION**

61970 The *recvfrom*( ) function shall receive a message from a connection-mode or connectionless-mode
61971 socket. It is normally used with connectionless-mode sockets because it permits the application
61972 to retrieve the source address of received data.

61973 The *recvfrom*( ) function takes the following arguments:

61974 *socket* Specifies the socket file descriptor.

61975 *buffer* Points to the buffer where the message should be stored.

61976 *length* Specifies the length in bytes of the buffer pointed to by the *buffer* argument.

61977 *flags* Specifies the type of message reception. Values of this argument are formed by
61978 logically OR'ing zero or more of the following values:

61979 MSG_PEEK Peeks at an incoming message. The data is treated as unread
61980 and the next *recvfrom*( ) or similar function shall still return
61981 this data.

61982 MSG_OOB Requests out-of-band data. The significance and semantics
61983 of out-of-band data are protocol-specific.

61984 MSG_WAITALL On SOCK_STREAM sockets this requests that the function
61985 block until the full amount of data can be returned. The
61986 function may return the smaller amount of data if the socket
61987 is a message-based socket, if a signal is caught, if the
61988 connection is terminated, if MSG_PEEK was specified, or if
61989 an error is pending for the socket.

61990 *address* A null pointer, or points to a **sockaddr** structure in which the sending address
61991 is to be stored. The length and format of the address depend on the address
61992 family of the socket.

61993 *address_len* Either a null pointer, if *address* is a null pointer, or a pointer to a **socklen_t**
61994 object which on input specifies the length of the supplied **sockaddr** structure,
61995 and on output specifies the length of the sending address.

61996 The *recvfrom*( ) function shall return the length of the message written to the buffer pointed to by
61997 RS the *buffer* argument. For message-based sockets, such as SOCK_RAW, SOCK_DGRAM, and
61998 SOCK_SEQPACKET, the entire message shall be read in a single operation. If a message is too
61999 long to fit in the supplied buffer, and MSG_PEEK is not set in the *flags* argument, the excess
62000 bytes shall be discarded. For stream-based sockets, such as SOCK_STREAM, message
62001 boundaries shall be ignored. In this case, data shall be returned to the user as soon as it becomes
62002 available, and no data shall be discarded.

62003 If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first
62004 message.

62005 Not all protocols provide the source address for messages. If the *address* argument is not a null

pointer and the protocol provides the source address of messages, the source address of the received message shall be stored in the **sockaddr** structure pointed to by the *address* argument, and the length of this address shall be stored in the object pointed to by the *address_len* argument.

If the actual length of the address is greater than the length of the supplied **sockaddr** structure, the stored address shall be truncated.

If the *address* argument is not a null pointer and the protocol does not provide the source address of messages, the value stored in the object pointed to by *address* is unspecified.

If no messages are available at the socket and O_NONBLOCK is not set on the socket's file descriptor, *recvfrom*( ) shall block until a message arrives or a timeout occurs (see SO_RCVTIMEO in Section 2.10.16, on page 554). If no messages are available at the socket and O_NONBLOCK is set on the socket's file descriptor, *recvfrom*( ) shall fail and set *errno* to [EAGAIN] or [EWOULDBLOCK].

RETURN VALUE

Upon successful completion, *recvfrom*( ) shall return the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, *recvfrom*( ) shall return 0. Otherwise, the function shall return –1 and set *errno* to indicate the error.

ERRORS

The *recvfrom*( ) function shall fail if:

[EAGAIN] or [EWOULDBLOCK]
: The socket's file descriptor is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket's file descriptor is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data. See also SO_RCVTIMEO in Section 2.10.16 (on page 554).

[EBADF]
: The *socket* argument is not a valid file descriptor.

[ECONNRESET]
: A connection was forcibly closed by a peer.

[EINTR]
: A signal interrupted *recvfrom*( ) before any data was available.

[EINVAL]
: The MSG_OOB flag is set and no out-of-band data is available.

[ENOTCONN]
: A receive is attempted on a connection-mode socket that is not connected.

[ENOTSOCK]
: The *socket* argument does not refer to a socket.

[EOPNOTSUPP]
: The specified flags are not supported for this socket type.

[ETIMEDOUT]
: The connection timed out during connection establishment, or due to a transmission timeout on active connection.

The *recvfrom*( ) function may fail if:

[EIO]
: An I/O error occurred while reading from or writing to the file system.

[ENOBUFS]
: Insufficient resources were available in the system to perform the operation.

[ENOMEM]
: Insufficient memory was available to fulfill the request.

EXAMPLES

None.

APPLICATION USAGE

The *select*( ) and *poll*( ) functions can be used to determine when data is available to be received.

For AF_UNIX sockets, it is recommended that *address* points to a buffer of length greater than `sizeof(struct sockaddr_un)` which has been initialized with null bytes. That way, even if the implementation supports the use of all bytes of *sun_path* without a terminating null byte, the larger buffer guarantees that the *sun_path* member can then be passed to other interfaces that expect a null-terminated string. If no truncation occurred based on the input value of *address_len*, it is unspecified whether the returned *address_len* will be `sizeof(struct sockaddr_un)`, or merely a value at least as large as `offsetof(struct sockaddr_un, sun_path)` plus the number of non-null bytes stored in *sun_path*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

poll( ), *pselect*( ), *read*( ), *recv*( ), *recvmsg*( ), *send*( ), *sendmsg*( ), *sendto*( ), *setsockopt*( ), *shutdown*( ), *socket*( ), *write*( )

XBD **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0503 [464] is applied.

Issue 8

Austin Group Defect 561 is applied, adding a paragraph about *sun_path* to APPLICATION USAGE.

Austin Group Defect 1429 is applied, clarifying the behavior on timeout by adding references to Section 2.10.16 (on page 554).

Austin Group Defect 1565 is applied, changing the description of *address_len*.

62075 **NAME**

62076 recvmsg — receive a message from a socket

62077 **SYNOPSIS**

62078
```
#include <sys/socket.h>
```

62079
```
ssize_t recvmsg(int socket, struct msghdr *message, int flags);
```

62080 **DESCRIPTION**

62081 The *recvmsg*( ) function shall receive a message from a connection-mode or connectionless-mode
62082 socket. It is normally used with connectionless-mode sockets because it permits the application
62083 to retrieve the source address of received data.

62084 The *recvmsg*( ) function takes the following arguments:

62085 *socket* Specifies the socket file descriptor.

62086 *message* Points to a **msghdr** structure, containing both the buffer to store the source
62087 address and the buffers for the incoming message. The length and format of
62088 the address depend on the address family of the socket. The *msg_flags* member
62089 is ignored on input, but may contain meaningful values on output.

62090 *flags* Specifies the type of message reception. Values of this argument are formed by
62091 logically OR'ing zero or more of the following values:

62092 MSG_OOB Requests out-of-band data. The significance and semantics
62093 of out-of-band data are protocol-specific.

62094 MSG_PEEK Peeks at the incoming message.

62095 MSG_WAITALL On SOCK_STREAM sockets this requests that the function
62096 block until the full amount of data can be returned. The
62097 function may return the smaller amount of data if the socket
62098 is a message-based socket, if a signal is caught, if the
62099 connection is terminated, if MSG_PEEK was specified, or if
62100 an error is pending for the socket.

62101 MSG_CMSG_CLOEXEC
62102 On sockets that permit a *cmsg_type* of SCM_RIGHTS in the
62103 *msg_control* ancillary data as a means of copying file
62104 descriptors into the process, the file descriptors shall be
62105 created with the FD_CLOEXEC flag atomically set.

62106 MSG_CMSG_CLOFORK
62107 On sockets that permit a *cmsg_type* of SCM_RIGHTS in the
62108 *msg_control* ancillary data as a means of copying file
62109 descriptors into the process, the file descriptors shall be
62110 created with the FD_CLOFORK flag atomically set.

62111 The *recvmsg*( ) function shall receive messages from unconnected or connected sockets and shall
62112 return the length of the message.

62113 The *recvmsg*( ) function shall return the total length of the message. For message-based sockets,
62114 such as SOCK_DGRAM and SOCK_SEQPACKET, the entire message shall be read in a single
62115 operation. If a message is too long to fit in the supplied buffers, and MSG_PEEK is not set in the
62116 *flags* argument, the excess bytes shall be discarded, and MSG_TRUNC shall be set in the
62117 *msg_flags* member of the **msghdr** structure. For stream-based sockets, such as SOCK_STREAM,
62118 message boundaries shall be ignored. In this case, data shall be returned to the user as soon as it
62119 becomes available, and no data shall be discarded.

If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first message.

If no messages are available at the socket and O_NONBLOCK is not set on the socket's file descriptor, *recvmsg*( ) shall block until a message arrives or a timeout occurs (see SO_RCVTIMEO in Section 2.10.16, on page 554). If no messages are available at the socket and O_NONBLOCK is set on the socket's file descriptor, the *recvmsg*( ) function shall fail and set *errno* to [EAGAIN] or [EWOULDBLOCK].

In the **msghdr** structure, the *msg_name* member may be a null pointer if the source address is not required. Otherwise, if the socket is unconnected, the *msg_name* member points to a **sockaddr** structure in which the source address is to be stored, and the *msg_namelen* member on input specifies the length of the supplied **sockaddr** structure and on output specifies the length of the source address. If the actual length of the address is greater than the length of the supplied **sockaddr** structure, the stored address shall be truncated. If the socket is connected, the *msg_name* and *msg_namelen* members shall be ignored. The *msg_iov* and *msg_iovlen* fields are used to specify where the received data shall be stored. The *msg_iov* member points to an array of **iovec** structures; the *msg_iovlen* member shall be set to the dimension of this array. In each **iovec** structure, the *iov_base* field specifies a storage area and the *iov_len* field gives its size in bytes. Each storage area indicated by *msg_iov* is filled with received data in turn until all of the received data is stored or all of the areas have been filled.

Upon successful completion, the *msg_flags* member of the message header shall be the bitwise-inclusive OR of all of the following flags that indicate conditions detected for the received message:

MSG_EOR End-of-record was received (if supported by the protocol).

MSG_OOB Out-of-band data was received.

MSG_TRUNC Normal data was truncated.

MSG_CTRUNC Control data was truncated.

RETURN VALUE

Upon successful completion, *recvmsg*( ) shall return the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, *recvmsg*( ) shall return 0. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *recvmsg*( ) function shall fail if:

[EAGAIN] or [EWOULDBLOCK]
The socket's file descriptor is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket's file descriptor is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data. See also SO_RCVTIMEO in Section 2.10.16 (on page 554).

[EBADF] The *socket* argument is not a valid open file descriptor.

[ECONNRESET] A connection was forcibly closed by a peer.

[EINTR] This function was interrupted by a signal before any data was available.

[EINVAL] The sum of the *iov_len* values overflows a **ssize_t**, or the MSG_OOB flag is set and no out-of-band data is available.

62163 [EMSGSIZE] The *msg_iovlen* member of the **msghdr** structure pointed to by *message* is less
62164 than or equal to 0, or is greater than {IOV_MAX}.

62165 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

62166 [ENOTSOCK] The *socket* argument does not refer to a socket.

62167 [EOPNOTSUPP] The specified flags are not supported for this socket type.

62168 [ETIMEDOUT] The connection timed out during connection establishment, or due to a
62169 transmission timeout on active connection.

62170 The *recvmsg*( ) function may fail if:

62171 [EIO] An I/O error occurred while reading from or writing to the file system.

62172 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

62173 [ENOMEM] Insufficient memory was available to fulfill the request.

62174 **EXAMPLES**

62175 None.

62176 **APPLICATION USAGE**

62177 The *select*( ) and *poll*( ) functions can be used to determine when data is available to be received.

62178 **RATIONALE**

62179 The use of the MSG_CMSG_CLOEXEC and MSG_CMSG_CLOFORK flags to *recvmsg*( ) when
62180 using SCM_RIGHTS to receive file descriptors via ancillary data is necessary to avoid a data race
62181 in multi-threaded applications. Without MSG_CMSG_CLOFORK, a file descriptor is leaked into
62182 a child process created by one thread in the window between another thread calling *recvmsg*( )
62183 and using *fcntl*( ) to set the FD_CLOFORK flag. Without MSG_CMSG_CLOEXEC, a file
62184 descriptor intentionally inherited by child processes is similarly leaked into an executed
62185 program if FD_CLOEXEC is not set atomically.

62186 **FUTURE DIRECTIONS**

62187 None.

62188 **SEE ALSO**

62189 *poll*( ), *pselect*( ), *recv*( ), *recvfrom*( ), *send*( ), *sendmsg*( ), *sendto*( ), *setsockopt*( ), *shutdown*( ), *socket*( )

62190 XBD **<sys/socket.h>**

62191 **CHANGE HISTORY**

62192 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

62193 **Issue 7**

62194 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0504 [464] is applied.

62195 **Issue 8**

62196 Austin Group Defects 411 and 1318 are applied, adding MSG_CMSG_CLOEXEC and
62197 MSG_CMSG_CLOFORK.

62198 Austin Group Defect 1429 is applied, clarifying the behavior on timeout by adding references to
62199 Section 2.10.16 (on page 554).

62200 Austin Group Defect 1565 is applied, changing the description of *msg_namelen*.

62201 **NAME**

62202 regcomp, regerror, regexec, regfree — regular expression matching

62203 **SYNOPSIS**

62204
```
#include <regex.h>
```

62205
```
int regcomp(regex_t *restrict preg, const char *restrict pattern,
    int cflags);
```

62207
```
size_t regerror(int errcode, const regex_t *restrict preg,
    char *restrict errbuf, size_t errbuf_size);
```

62209
```
int regexec(const regex_t *restrict preg, const char *restrict string,
    size_t nmatch, regmatch_t pmatch[restrict], int eflags);
```

62211
```
void regfree(regex_t *preg);
```

62212 **DESCRIPTION**

62213 These functions interpret *basic* and *extended* regular expressions as described in XBD Chapter 9
62214 (on page 179).

62215 The **regex_t** structure is defined in **<regex.h>** and contains at least the following member:

62216

Member Type	Member Name	Description
size_t	*re_nsub*	Number of parenthesized subexpressions.

62219 The **regmatch_t** structure is defined in **<regex.h>** and contains at least the following members:

62220

Member Type	Member Name	Description
regoff_t	*rm_so*	Byte offset from start of *string* to start of substring.
regoff_t	*rm_eo*	Byte offset from start of *string* of the first character after the end of substring.

62225 The *regcomp*( ) function shall compile the regular expression contained in the string pointed to by
62226 the *pattern* argument and place the results in the structure pointed to by *preg*. The *cflags*
62227 argument is the bitwise-inclusive OR of zero or more of the following flags, which are defined in
62228 the **<regex.h>** header:

62229 REG_EXTENDED Use Extended Regular Expressions.

62230 REG_ICASE Perform matching in a case-insensitive manner (see XBD Section 9.2, on
62231 page 180).

62232 REG_MINIMAL Change the matching behavior for duplication symbols to the leftmost
62233 shortest possible match, and invert the behavior of the repetition modifier
62234 `'?'` (<question-mark>) to match the longest possible match instead of the
62235 shortest. Only applicable to REG_EXTENDED regular expressions.

62236 REG_NOSUB Report only success/fail in *regexec*( ).

62237 REG_NEWLINE Change the handling of <newline> characters, as described in the text.

62238 The default regular expression type for *pattern* is a Basic Regular Expression. The application can
62239 specify Extended Regular Expressions using the REG_EXTENDED *cflags* flag.

62240 If the REG_NOSUB flag was not set in *cflags*, then *regcomp*( ) shall set *re_nsub* to the number of
62241 parenthesized subexpressions (delimited by `"\(\)"` in basic regular expressions or `"()"` in
62242 extended regular expressions) found in *pattern*.

62243 The *regexec*( ) function compares the null-terminated string specified by *string* with the compiled

62244 regular expression *preg* initialized by a previous call to *regcomp*( ). If it finds a match, *regexec*( )
62245 shall return 0; otherwise, it shall return non-zero indicating either no match or an error. The
62246 *eflags* argument is the bitwise-inclusive OR of zero or more of the following flags, which are
62247 defined in the **<regex.h>** header:

62248 REG_NOTBOL The first character of the string pointed to by *string* is not the beginning of
62249 the line. Therefore, the <circumflex> character (`'^'`), when taken as a
62250 special character, shall not match the beginning of *string*.

62251 REG_NOTEOL The last character of the string pointed to by *string* is not the end of the
62252 line. Therefore, the <dollar-sign> (`'$'`), when taken as a special character,
62253 shall not match the end of *string*.

62254 If *nmatch* is 0 or REG_NOSUB was set in the *cflags* argument to *regcomp*( ), then *regexec*( ) shall
62255 ignore the *pmatch* argument. Otherwise, the application shall ensure that the *pmatch* argument
62256 points to an array with at least *nmatch* elements, and *regexec*( ) shall fill in the elements of that
62257 array with offsets of the substrings of *string* that correspond to the parenthesized subexpressions
62258 of *pattern*: *pmatch*[*i*].*rm_so* shall be the byte offset of the beginning and *pmatch*[*i*].*rm_eo* shall be
62259 one greater than the byte offset of the end of substring *i*. (Subexpression *i* begins at the *i*th
62260 matched open parenthesis, counting from 1.) Offsets in *pmatch*[0] identify the substring that
62261 corresponds to the entire regular expression. Unused elements of *pmatch* up to *pmatch*[*nmatch*–1]
62262 shall be filled with –1. If there are more than *nmatch* subexpressions in *pattern* (*pattern* itself
62263 counts as a subexpression), then *regexec*( ) shall still do the match, but shall record only the first
62264 *nmatch* substrings.

62265 When matching a basic or extended regular expression, any given parenthesized subexpression
62266 of *pattern* might participate in the match of several different substrings of *string*, or it might not
62267 match any substring even though the pattern as a whole did match. The following rules shall be
62268 used to determine which substrings to report in *pmatch* when matching regular expressions:

62269 1. If subexpression *i* in a regular expression is not contained within another subexpression,
62270 and it participated in the match several times, then the byte offsets in *pmatch*[*i*] shall
62271 delimit the last such match.

62272 2. If subexpression *i* is not contained within another subexpression, and it did not
62273 participate in an otherwise successful match, the byte offsets in *pmatch*[*i*] shall be –1. A
62274 subexpression does not participate in the match when:

62275 `'*'` or `"\{\}"` appears immediately after the subexpression in a basic regular
62276 expression, or `'*'`, `'?'`, or `"{}"` appears immediately after the subexpression in
62277 an extended regular expression, and the subexpression did not match (matched 0
62278 times)

62279 or:

62280 `'|'` is used in an extended regular expression to select this subexpression or
62281 another, and the other subexpression matched.

62282 3. If subexpression *i* is contained within another subexpression *j*, and *i* is not contained
62283 within any other subexpression that is contained within *j*, and a match of subexpression *j*
62284 is reported in *pmatch*[*j*], then the match or non-match of subexpression *i* reported in
62285 *pmatch*[*i*] shall be as described in 1. and 2. above, but within the substring reported in
62286 *pmatch*[*j*] rather than the whole string. The offsets in *pmatch*[*i*] are still relative to the start
62287 of *string*.

4. If subexpression *i* is contained in subexpression *j*, and the byte offsets in *pmatch*[*j*] are –1, then the pointers in *pmatch*[*i*] shall also be –1.
5. If subexpression *i* matched a zero-length string, then both byte offsets in *pmatch*[*i*] shall be the byte offset of the character or null terminator immediately following the zero-length string.

If, when *regexec*() is called, the locale is different from when the regular expression was compiled, the result is undefined.

If REG_NEWLINE is not set in *cflags*, then a <newline> in *pattern* or *string* shall be treated as an ordinary character. If REG_NEWLINE is set, then <newline> shall be treated as an ordinary character except as follows:

1. A <newline> in *string* shall not be matched by a <period> outside a bracket expression or by any form of a non-matching list (see XBD Chapter 9, on page 179).
2. A <circumflex> (`'^'`) in *pattern*, when used to specify expression anchoring (see XBD Section 9.3.8, on page 186), shall match the zero-length string immediately after a <newline> in *string*, regardless of the setting of REG_NOTBOL.
3. A <dollar-sign> (`'$'`) in *pattern*, when used to specify expression anchoring, shall match the zero-length string immediately before a <newline> in *string*, regardless of the setting of REG_NOTEOL.

The *regfree*() function shall free any memory allocated by *regcomp*() associated with *preg*. The *regfree*() function shall not modify *errno* if *preg* was previously returned by *regcomp*() and not yet freed.

The following constants are defined as the minimum set of error return values, although other errors listed as implementation extensions in **<regex.h>** are possible:

REG_BADBR	Content of `"\{\}"` invalid: not a number, number too large, more than two numbers, first larger than second.
REG_BADPAT	Invalid regular expression.
REG_BADRPT	`'?'`, `'*'`, or `'+'` not preceded by valid regular expression.
REG_EBRACE	`"\{\}"` imbalance.
REG_EBRACK	`"[]"` imbalance.
REG_ECOLLATE	Invalid collating element referenced.
REG_ECTYPE	Invalid character class type referenced.
REG_EESCAPE	Trailing <backslash> character in pattern.
REG_EPAREN	`"\(\)"` or `"()"` imbalance.
REG_ERANGE	Invalid endpoint in range expression.
REG_ESPACE	Out of memory.
REG_ESUBREG	Number in `"\digit"` invalid or in error.
REG_NOMATCH	*regexec*() failed to match.

If more than one error occurs in processing a function call, any one of the possible constants may be returned, as the order of detection is unspecified.

The *regerror*() function provides a mapping from error codes returned by *regcomp*() and

regexec( ) to unspecified printable strings. It generates a string corresponding to the value of the *errcode* argument, which the application shall ensure is the last non-zero value returned by *regcomp*( ) or *regexec*( ) with the given value of *preg*. If *errcode* is not such a value, the content of the generated string is unspecified.

If *preg* is a null pointer, but *errcode* is a value returned by a previous call to *regexec*( ) or *regcomp*( ), the *regerror*( ) still generates an error string corresponding to the value of *errcode*, but it might not be as detailed under some implementations.

If the *errbuf_size* argument is not 0, *regerror*( ) shall place the generated string into the buffer of size *errbuf_size* bytes pointed to by *errbuf*. If the string (including the terminating null) cannot fit in the buffer, *regerror*( ) shall truncate the string and null-terminate the result.

If *errbuf_size* is 0, *regerror*( ) shall ignore the *errbuf* argument, and return the size of the buffer needed to hold the generated string.

If the *preg* argument to *regexec*( ) or *regfree*( ) is not a compiled regular expression returned by *regcomp*( ), the result is undefined. A *preg* is no longer treated as a compiled regular expression after it is given to *regfree*( ).

RETURN VALUE

Upon successful completion, the *regcomp*( ) function shall return 0. Otherwise, it shall return an integer value indicating an error as described in **<regex.h>**, and the content of *preg* is undefined. If a code is returned, the interpretation shall be as given in **<regex.h>**.

If *regcomp*( ) detects an invalid RE, it may return REG_BADPAT, or it may return one of the error codes that more precisely describes the error.

Upon successful completion, the *regexec*( ) function shall return 0. Otherwise, it shall return REG_NOMATCH to indicate no match.

Upon successful completion, the *regerror*( ) function shall return the number of bytes needed to hold the entire generated string, including the null termination. If the return value is greater than *errbuf_size*, the string returned in the buffer pointed to by *errbuf* has been truncated.

The *regfree*( ) function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

```
#include <regex.h>

/*
 * Match string against the extended regular expression in
 * pattern, treating errors as no match.
 *
 * Return 1 for match, 0 for no match.
 */

int
match(const char *string, char *pattern)
{
    int    status;
    regex_t    re;

    if (regcomp(&re, pattern, REG_EXTENDED|REG_NOSUB) != 0) {
        return(0);      /* Report error. */
```



```
        }
        status = regexec(&re, string, (size_t) 0, NULL, 0);
        regfree(&re);
        if (status != 0) {
            return(0);      /* Report error. */
        }
        return(1);
    }
```

The following demonstrates how the REG_NOTBOL flag could be used with *regexec*( ) to find all substrings in a line that match a pattern supplied by a user. (For simplicity of the example, very little error checking is done.)

```
(void) regcomp (&re, pattern, 0);
/* This call to regexec() finds the first match on the line. */
error = regexec (&re, &buffer[0], 1, &pm, 0);
while (error == 0) {  /* While matches found. */
    /* Substring found between pm.rm_so and pm.rm_eo. */
    /* This call to regexec() finds the next match. */
    error = regexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
}
```

APPLICATION USAGE

An application could use:

```
regerror(code,preg,(char *)NULL,(size_t)0)
```

to find out how big a buffer is needed for the generated string, *malloc*( ) a buffer to hold the string, and then call *regerror*( ) again to get the string. Alternatively, it could allocate a fixed, static buffer that is big enough to hold most strings, and then use *malloc*( ) to allocate a larger buffer if it finds that this is too small.

To match a pattern as described in XCU Section 2.14 (on page 2523), use the *fnmatch*( ) function.

RATIONALE

The *regexec*( ) function must fill in all *nmatch* elements of *pmatch*, where *nmatch* and *pmatch* are supplied by the application, even if some elements of *pmatch* do not correspond to subexpressions in *pattern*. The application developer should note that there is probably no reason for using a value of *nmatch* that is larger than *preg*–>*re_nsub*+1.

The REG_NEWLINE flag supports a use of RE matching that is needed in some applications like text editors. In such applications, the user supplies an RE asking the application to find a line that matches the given expression. An anchor in such an RE anchors at the beginning or end of any line. Such an application can pass a sequence of <newline>-separated lines to *regexec*( ) as a single long string and specify REG_NEWLINE to *regcomp*( ) to get the desired behavior. The application must ensure that there are no explicit <newline> characters in *pattern* if it wants to ensure that any match occurs entirely within a single line.

The REG_NEWLINE flag affects the behavior of *regexec*( ), but it is in the *cflags* parameter to *regcomp*( ) to allow flexibility of implementation. Some implementations will want to generate the same compiled RE in *regcomp*( ) regardless of the setting of REG_NEWLINE and have *regexec*( ) handle anchors differently based on the setting of the flag. Other implementations will generate different compiled REs based on the REG_NEWLINE.

The REG_ICASE flag supports the operations taken by the *grep* **–i** option and the historical implementations of *ex* and *vi*. Including this flag will make it easier for application code to be written that does the same thing as these utilities.

The substrings reported in *pmatch*[ ] are defined using offsets from the start of the string rather than pointers. This allows type-safe access to both constant and non-constant strings.

The type **regoff_t** is used for the elements of *pmatch*[ ] to ensure that the application can represent large arrays in memory (important for an application conforming to the Shell and Utilities volume of POSIX.1-2024).

The 1992 edition of this standard required **regoff_t** to be at least as wide as **off_t**, to facilitate future extensions in which the string to be searched is taken from a file. However, these future extensions have not appeared. The requirement rules out popular implementations with 32-bit **regoff_t** and 64-bit **off_t**, so it has been removed.

The standard developers rejected the inclusion of a *regsub*( ) function that would be used to do substitutions for a matched RE. While such a routine would be useful to some applications, its utility would be much more limited than the matching function described here. Both RE parsing and substitution are possible to implement without support other than that required by the ISO C standard, but matching is much more complex than substituting. The only difficult part of substitution, given the information supplied by *regexec*( ), is finding the next character in a string when there can be multi-byte characters. That is a much larger issue, and one that needs a more general solution.

The *errno* variable has not been used for error returns to avoid filling the *errno* name space for this feature.

The interface is defined so that the matched substrings *rm_sp* and *rm_ep* are in a separate **regmatch_t** structure instead of in **regex_t**. This allows a single compiled RE to be used simultaneously in several contexts; in *main*( ) and a signal handler, perhaps, or in multiple threads of lightweight processes. (The *preg* argument to *regexec*( ) is declared with type **const**, so the implementation is not permitted to use the structure to store intermediate results.) It also allows an application to request an arbitrary number of substrings from an RE. The number of subexpressions in the RE is reported in *re_nsub* in *preg*. With this change to *regexec*( ), consideration was given to dropping the REG_NOSUB flag since the user can now specify this with a zero *nmatch* argument to *regexec*( ). However, keeping REG_NOSUB allows an implementation to use a different (perhaps more efficient) algorithm if it knows in *regcomp*( ) that no subexpressions need be reported. The implementation is only required to fill in *pmatch* if *nmatch* is not zero and if REG_NOSUB is not specified. Note that the **size_t** type, as defined in the ISO C standard, is unsigned, so the description of *regexec*( ) does not need to address negative values of *nmatch*.

REG_NOTBOL was added to allow an application to do repeated searches for the same pattern in a line. If the pattern contains a <circumflex> character that should match the beginning of a line, then the pattern should only match when matched against the beginning of the line. Without the REG_NOTBOL flag, the application could rewrite the expression for subsequent matches, but in the general case this would require parsing the expression. The need for REG_NOTEOL is not as clear; it was added for symmetry.

The addition of the *regerror*( ) function addresses the historical need for conforming application programs to have access to error information more than ``Function failed to compile/match your RE for unknown reasons''.

This interface provides for two different methods of dealing with error conditions. The specific error codes (REG_EBRACE, for example), defined in **<regex.h>**, allow an application to recover from an error if it is so able. Many applications, especially those that use patterns supplied by a user, will not try to deal with specific error cases, but will just use *regerror*( ) to obtain a human-readable error message to present to the user.

The *regerror*( ) function uses a scheme similar to *confstr*( ) to deal with the problem of allocating memory to hold the generated string. The scheme used by *strerror*( ) in the ISO C standard was considered unacceptable since it creates difficulties for multi-threaded applications.

The *preg* argument is provided to *regerror*( ) to allow an implementation to generate a more descriptive message than would be possible with *errcode* alone. An implementation might, for example, save the character offset of the offending character of the pattern in a field of *preg*, and then include that in the generated message string. The implementation may also ignore *preg*.

A REG_FILENAME flag was considered, but omitted. This flag caused *regexec*( ) to match patterns as described in XCU Section 2.14 (on page 2523) instead of REs. This service is now provided by the *fnmatch*( ) function.

Notice that there is a difference in philosophy between the ISO POSIX-2: 1993 standard and POSIX.1-2024 in how to handle a ``bad'' regular expression. The ISO POSIX-2: 1993 standard says that many bad constructs ``produce undefined results'', or that ``the interpretation is undefined''. POSIX.1-2024, however, says that the interpretation of such REs is unspecified. The term ``undefined'' means that the action by the application is an error, of similar severity to passing a bad pointer to a function.

The *regcomp*( ) and *regexec*( ) functions are required to accept any null-terminated string as the *pattern* argument. If the meaning of the string is ``undefined'', the behavior of the function is ``unspecified''. POSIX.1-2024 does not specify how the functions will interpret the pattern; they might return error codes, or they might do pattern matching in some completely unexpected way, but they should not do something like abort the process.

FUTURE DIRECTIONS

None.

SEE ALSO

fnmatch( ), *glob*( )

XBD Chapter 9 (on page 179), **<regex.h>**, **<sys/types.h>**

XCU Section 2.14 (on page 2523)

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 5

Moved from POSIX2 C-language Binding to BASE.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

The normative text is updated to avoid use of the term ``must'' for application requirements.

The REG_ENOSYS constant is removed.

The **restrict** keyword is added to the *regcomp*( ), *regerror*( ), and *regexec*( ) prototypes for alignment with the ISO/IEC 9899: 1999 standard.

62508 **Issue 7**

62509 Austin Group Interpretation 1003.1-2001 #134 is applied, clarifying that if more than one error
62510 occurs in processing a function call, any one of the possible constants may be returned.

62511 SD5-XBD-ERN-60 is applied.

62512 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0505 [305] is applied.

62513 **Issue 8**

62514 Austin Group Defect 385 is applied, adding a requirement that *regfree*( ) does not modify *errno*
62515 when passed a pointer to a **regex_t** that can be freed.

62516 Austin Group Defects 793 and 1329 are applied, adding REG_MINIMAL.

62517 Austin Group Defect 1031 is applied, changing the description of REG_ICASE.

62518 **NAME**

62519 remainder, remainderf, remainderl — remainder function

62520 **SYNOPSIS**

62521
```
#include <math.h>
```

62522
```
double remainder(double x, double y);
```
62523
```
float remainderf(float x, float y);
```
62524
```
long double remainderl(long double x, long double y);
```

62525 **DESCRIPTION**

62526 CX The functionality described on this reference page is aligned with the ISO C standard. Any
62527 conflict between the requirements described here and the ISO C standard is unintentional. This
62528 volume of POSIX.1-2024 defers to the ISO C standard.

62529 These functions shall return the floating-point remainder $r=x-ny$ when y is non-zero. The value
62530 n is the integral value nearest the exact value x/y. When $|\, n-x/y \,|=½$, the value n is chosen to
62531 be even.

62532 The behavior of *remainder*( ) shall be independent of the rounding mode.

62533 **RETURN VALUE**

62534 Upon successful completion, these functions shall return the floating-point remainder $r=x-ny$
62535 when y is non-zero.

62536 MX When subnormal results are supported, the returned value shall be exact.

62537 On systems that do not support the IEC 60559 Floating-Point option, if y is zero, it is
62538 implementation-defined whether a domain error occurs or zero is returned.

62539 MX If x or y is NaN, a NaN shall be returned.

62540 If x is infinite or y is 0 and the other is non-NaN, a domain error shall occur, and a NaN shall be
62541 returned.

62542 **ERRORS**

62543 These functions shall fail if:

62544 MX Domain Error The x argument is ±Inf, or the y argument is ±0 and the other argument is non-
62545 NaN.

62546 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
62547 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
62548 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
62549 shall be raised.

62550 These functions may fail if:

62551 Domain Error The y argument is zero.

62552 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
62553 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
62554 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
62555 shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

abs( ), *div*( ), *feclearexcept*( ), *fetestexcept*( ), *ldiv*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The *remainder*( ) function is no longer marked as an extension.

The *remainderf*( ) and *remainderl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

Issue 7

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #55 (SD5-XSH-ERN-82) is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0506 [320] is applied.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018 standard.

NAME

remove — remove a file

SYNOPSIS

```
#include <stdio.h>

int remove(const char *path);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *remove*( ) function shall cause the file named by the pathname pointed to by *path* to be no longer accessible by that name. A subsequent attempt to open that file using that name shall fail, unless it is created anew.

CX If *path* does not name a directory, *remove*(*path*) shall be equivalent to *unlink*(*path*).

If *path* names a directory, *remove*(*path*) shall be equivalent to *rmdir*(*path*).

RETURN VALUE

CX Refer to *rmdir*( ) or *unlink*( ).

ERRORS

CX Refer to *rmdir*( ) or *unlink*( ).

EXAMPLES

Removing Access to a File

The following example shows how to remove access to a file named **/home/cnd/old_mods**.

```
#include <stdio.h>
int status;
...
status = remove("/home/cnd/old_mods");
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

rmdir( ), *unlink*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ISO C standard.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION, RETURN VALUE, and ERRORS sections are updated so that if *path* is not a directory, *remove*( ) is equivalent to *unlink*( ), and if it is a directory, it is equivalent to *rmdir*( ).

62630 **NAME**

62631 remque — remove an element from a queue

62632 **SYNOPSIS**

62633 XSI
```
#include <search.h>

void remque(void *element);
```

62635 **DESCRIPTION**

62636 Refer to *insque( )*.

NAME

remquo, remquof, remquol — remainder functions

SYNOPSIS

```
#include <math.h>

double remquo(double x, double y, int *quo);
float remquof(float x, float y, int *quo);
long double remquol(long double x, long double y, int *quo);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *remquo*( ), *remquof*( ), and *remquol*( ) functions shall compute the same remainder as the *remainder*( ), *remainderf*( ), and *remainderl*( ) functions, respectively. In the object pointed to by *quo*, they store a value whose sign is the sign of x/y and whose magnitude is congruent modulo 2^n to the magnitude of the integral quotient of x/y, where n is an implementation-defined integer greater than or equal to 3. If y is zero, the value stored in the object pointed to by *quo* is unspecified.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

These functions shall return x REM y.

MX When subnormal results are supported, the returned value shall be exact.

If NaN is supported and a NaN is returned, the value stored in the object pointed to by *quo* is unspecified.

On systems that do not support the IEC 60559 Floating-Point option, if y is zero, it is implementation-defined whether a domain error occurs or zero is returned.

MX If x or y is NaN, a NaN shall be returned.

If x is ±Inf or y is zero and the other argument is non-NaN, a domain error shall occur, and a NaN shall be returned.

ERRORS

These functions shall fail if:

MX

Domain Error — The x argument is ±Inf, or the y argument is ±0 and the other argument is non-NaN.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Domain Error — The y argument is zero.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception

shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions are intended for implementing argument reductions which can exploit a few low-order bits of the quotient. Note that *x* may be so large in magnitude relative to *y* that an exact representation of the quotient is not practical.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept( )*, *remainder( )*

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #56 (SD5-XSH-ERN-83) is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0507 [320] is applied.

Issue 8

Austin Group Defect 713 is applied, clarifying the behavior when a NaN is returned.

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

NAME

rename, renameat — rename file

SYNOPSIS

```
#include <stdio.h>

int rename(const char *old, const char *new);
```

OH CX
```
#include <fcntl.h>
```

CX
```
int renameat(int oldfd, const char *old, int newfd,
    const char *new);
```

DESCRIPTION

CX For *rename*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *rename*( ) function shall change the name of a file. The *old* argument points to the pathname CX of the file to be renamed. The *new* argument points to the new pathname of the file. If the *new* argument does not resolve to an existing directory entry for a file of type directory and the *new* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters after all symbolic links have been processed, *rename*( ) shall fail.

If either the *old* or *new* argument names a symbolic link, *rename*( ) shall operate on the symbolic link itself, and shall not resolve the last component of the argument. If the *old* argument and the *new* argument resolve to either the same existing directory entry or different directory entries for the same existing file, *rename*( ) shall return successfully and perform no other action.

If the *old* argument names a file that is not a directory and the *new* argument names a directory, or *old* names a directory and *new* names a file that is not a directory, or *new* names a directory that is not empty, *rename*( ) shall fail. Otherwise, if the directory entry named by *new* exists, it shall be removed and *old* renamed to *new*. In this case, a directory entry named *new* shall remain visible to other threads throughout the renaming operation and refer either to the file referred to by *new* or *old* before the operation began.

If either *pathname* argument refers to a path whose final component is either dot or dot-dot, *rename*( ) shall fail.

If the *old* argument points to a pathname of a symbolic link, the symbolic link shall be renamed. If the *new* argument points to a pathname of a symbolic link, the symbolic link shall be removed.

The *old* pathname shall not name an ancestor directory of the *new* pathname. Write access permission is required for the directory containing *old* and the directory containing *new*. If the *old* argument points to the pathname of a directory, write access permission may be required for the directory named by *old*, and, if it exists, the directory named by *new*.

If the *new* argument names an existing file and the file's link count becomes 0 when it is removed and no process has the file open, the space occupied by the file shall be freed and the file shall no longer be accessible. If one or more processes have the file open when the last link is removed, the link shall be removed before *rename*( ) returns, but the removal of the file contents shall be postponed until all references to the file are closed.

Upon successful completion, *rename*( ) shall mark for update the last data modification and last file status change timestamps of the parent directory of each file.

If the *rename*( ) function fails for any reason other than [EIO], any file named by *new* shall be unaffected.

The *renameat*( ) function shall be equivalent to the *rename*( ) function except in the case where either *old* or *new* specifies a relative path. If *old* is a relative path, the file to be renamed is located relative to the directory associated with the file descriptor *oldfd* instead of the current working directory. If *new* is a relative path, the same happens only relative to the directory associated with *newfd*. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

If *renameat*( ) is passed the special value AT_FDCWD in the *oldfd* or *newfd* parameter, the current working directory shall be used in the determination of the file for the respective *path* parameter.

RETURN VALUE

CX Upon successful completion, the *rename*( ) function shall return 0. Otherwise, it shall return –1, *errno* shall be set to indicate the error, and neither the file named by *old* nor the file named by *new* shall be changed or created.

CX Upon successful completion, the *renameat*( ) function shall return 0. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

CX The *rename*( ) and *renameat*( ) functions shall fail if:

CX **[EACCES]** A component of either path prefix denies search permission; or one of the directories containing *old* or *new* denies write permissions; or, write permission is required and is denied for a directory pointed to by the *old* or *new* arguments.

CX **[EBUSY]** The directory named by *old* or *new* is currently in use by the system or another process, and the implementation considers this an error.

CX **[EEXIST] or [ENOTEMPTY]**
The *new* argument names a directory that is not empty.

CX **[EILSEQ]** The last pathname component of the new pathname is not a portable filename, and cannot be created in the target directory.

CX **[EINVAL]** The *old* pathname names an ancestor directory of the *new* pathname, or either pathname argument contains a final component that is dot or dot-dot.

CX **[EIO]** A physical I/O error has occurred.

CX **[EISDIR]** The *new* argument points to a directory and the *old* argument points to a file that is not a directory.

CX **[ELOOP]** A loop exists in symbolic links encountered during resolution of the *old* or *new* argument.

CX **[EMLINK]** The file named by *old* is a directory, and the link count of the parent directory of *new* would exceed {LINK_MAX}.

CX **[ENAMETOOLONG]**
The length of a component of a pathname is longer than {NAME_MAX}.

CX **[ENOENT]** The *old* argument does not name an existing file, a component of the path prefix of *new* does not exist, or either *old* or *new* points to an empty string.

62790 CX [ENOSPC] The directory that would contain *new* cannot be extended.

62791 CX [ENOTDIR] A component of either path prefix names an existing file that is neither a
62792 directory nor a symbolic link to a directory; or the *old* argument names a
62793 directory and the *new* argument names a non-directory file; or the *old*
62794 argument contains at least one non-<slash> character and ends with one or
62795 more trailing <slash> characters and the last pathname component names an
62796 existing file that is neither a directory nor a symbolic link to a directory; or the
62797 *old* argument names an existing non-directory file and the *new* argument
62798 names a nonexistent file, contains at least one non-<slash> character, and ends
62799 with one or more trailing <slash> characters; or the *new* argument names an
62800 existing non-directory file, contains at least one non-<slash> character, and
62801 ends with one or more trailing <slash> characters.

62802 XSI [EPERM] or [EACCES]
62803 The S_ISVTX flag is set on the directory containing the file referred to by *old*
62804 and the process does not satisfy the criteria specified in XBD Section 4.5 (on
62805 page 96) with respect to *old*; or *new* refers to an existing file, the S_ISVTX flag
62806 is set on the directory containing this file, and the process does not satisfy the
62807 criteria specified in XBD Section 4.5 with respect to this file.

62808 CX [EROFS] The requested operation requires writing in a directory on a read-only file
62809 system.

62810 CX [EXDEV] The file named by *old* and the directory in which the directory entry named by
62811 *new* is to be created or replaced are on different file systems and the
62812 implementation does not support hard links between file systems.

62813 CX In addition, the *renameat*( ) function shall fail if:

62814 [EACCES] The access mode of the open file description associated with *oldfd* or *newfd* is
62815 not O_SEARCH and the permissions of the directory underlying *oldfd* or
62816 *newfd*, respectively, do not permit directory searches.

62817 [EBADF] The *old* argument does not specify an absolute path and the *oldfd* argument is
62818 neither AT_FDCWD nor a valid file descriptor open for reading or searching,
62819 or the *new* argument does not specify an absolute path and the *newfd*
62820 argument is neither AT_FDCWD nor a valid file descriptor open for reading
62821 or searching.

62822 [ENOTDIR] The *old* or *new* argument is not an absolute path and *oldfd* or *newfd*,
62823 respectively, is a file descriptor associated with a non-directory file.

62824 CX The *rename*( ) and *renameat*( ) functions may fail if:

62825 CX [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
62826 resolution of the *old* or *new* argument.

62827 CX [ENAMETOOLONG]
62828 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
62829 symbolic link produced an intermediate result with a length that exceeds
62830 {PATH_MAX}.

62831 CX [ETXTBSY] The file named by *new* exists and is the last directory entry to a pure procedure
62832 (shared text) file that is being executed.

EXAMPLES

Renaming a File

The following example shows how to rename a file named **/home/cnd/mod1** to **/home/cnd/mod2**.

```
#include <stdio.h>

int status;
...
status = rename("/home/cnd/mod1", "/home/cnd/mod2");
```

APPLICATION USAGE

Some implementations mark for update the last file status change timestamp of renamed files and some do not. Applications which make use of the last file status change timestamp may behave differently with respect to renamed files unless they are designed to allow for either behavior.

RATIONALE

This *rename*( ) function is equivalent for regular files to that defined by the ISO C standard. Its inclusion here expands that definition to include actions on directories and specifies behavior when the *new* parameter names a file that already exists. That specification requires that the action of the function be atomic.

One of the reasons for introducing this function was to have a means of renaming directories while permitting implementations to prohibit the use of *link*( ) and *unlink*( ) with directories, thus constraining links to directories to those made by *mkdir*( ).

The specification that if *old* and *new* refer to the same file is intended to guarantee that:

```
rename("x", "x");
```

does not remove the file.

Renaming dot or dot-dot is prohibited in order to prevent cyclical file system paths.

See also the descriptions of [ENOTEMPTY] and [ENAMETOOLONG] in *rmdir*( ) and [EBUSY] in *unlink*( ). For a discussion of [EXDEV], see *link*( ).

The purpose of the *renameat*( ) function is to rename files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *rename*( ), resulting in unspecified behavior. By opening file descriptors for the source and target directories and using the *renameat*( ) function it can be guaranteed that that renamed file is located correctly and the resulting file is in the desired directory.

Implementations are encouraged to have *rename*( ) and *renameat*( ) report an [EILSEQ] error if the file named by *new* does not already exist and the last component of that pathname contains any bytes that have the encoded value of a <newline> character.

FUTURE DIRECTIONS

None.

SEE ALSO

link( ), *rmdir*( ), *symlink*( ), *unlink*( )

XBD Section 4.5 (on page 96), **<fcntl.h>**, **<stdio.h>**

62874 **CHANGE HISTORY**

62875 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

62876 **Issue 5**

62877 The [EBUSY] error is added to the optional part of the ERRORS section.

62878 **Issue 6**

62879 Extensions beyond the ISO C standard are marked.

62880 The following new requirements on POSIX implementations derive from alignment with the
62881 Single UNIX Specification:

62882 • The [EIO] mandatory error condition is added.

62883 • The [ELOOP] mandatory error condition is added.

62884 • A second [ENAMETOOLONG] is added as an optional error condition.

62885 • The [ETXTBSY] optional error condition is added.

62886 The following changes were made to align with the IEEE P1003.1a draft standard:

62887 • Details are added regarding the treatment of symbolic links.

62888 • The [ELOOP] optional error condition is added.

62889 The normative text is updated to avoid use of the term ``must'' for application requirements.

62890 **Issue 7**

62891 Austin Group Interpretation 1003.1-2001 #016 is applied, changing the definition of the
62892 [ENOTDIR] error.

62893 Austin Group Interpretation 1003.1-2001 #076 is applied, clarifying the behavior if the final
62894 component of a path is either dot or dot-dot, and adding the associated [EINVAL] error case.

62895 Austin Group Interpretation 1003.1-2001 #143 is applied.

62896 Austin Group Interpretation 1003.1-2001 #145 is applied, clarifying that the [ENOENT] error
62897 condition also applies to the case in which a component of *new* does not exist.

62898 Austin Group Interpretations 1003.1-2001 #174 and #181 are applied.

62899 The *renameat*( ) function is added from The Open Group Technical Standard, 2006, Extended API
62900 Set Part 2.

62901 Changes are made related to support for finegrained timestamps.

62902 Changes are made to allow a directory to be opened for searching.

62903 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0508 [324], XSH/TC1-2008/0509 [147],
62904 XSH/TC1-2008/0510 [379], XSH/TC1-2008/0511 [278], and XSH/TC1-2008/0512 [278] are
62905 applied.

62906 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0311 [873], XSH/TC2-2008/0312 [591],
62907 XSH/TC2-2008/0313 [716], XSH/TC2-2008/0314 [817], XSH/TC2-2008/0315 [817], and
62908 XSH/TC2-2008/0316 [591] are applied.

62909 **Issue 8**

62910 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of
62911 filenames containing any bytes that have the encoded value of a <newline> character.

62912 Austin Group Defect 293 is applied, adding the [EILSEQ] error.

62913 Austin Group Defect 1200 is applied, correcting the argument names in the [ELOOP] errors.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1380 is applied, changing text using the term ``link'' in line with its updated definition.

NAME

rewind — reset the file position indicator in a stream

SYNOPSIS

```
#include <stdio.h>

void rewind(FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The call:

```
rewind(stream)
```

shall be equivalent to:

```
(void) fseek(stream, 0L, SEEK_SET)
```

except that *rewind*( ) shall also ensure the error indicator is clear when the function returns.

CX Since *rewind*( ) does not return a value, an application wishing to detect errors should clear *errno*, then call *rewind*( ), and if *errno* is non-zero, assume an error has occurred.

RETURN VALUE

The *rewind*( ) function shall not return a value.

ERRORS

CX Refer to *fseek*( ) with the exception of [EINVAL] which does not apply.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fseek*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0513 [14] is applied.

Issue 8

Austin Group Defect 1414 is applied, clarifying that *rewind*( ) ensures the error indicator is clear when the function returns.

62957 **NAME**

62958 rewinddir — reset the position of a directory stream to the beginning of a directory

62959 **SYNOPSIS**

62960 `#include <dirent.h>`

62961 `void rewinddir(DIR *dirp);`

62962 **DESCRIPTION**

62963 The *rewinddir*( ) function shall reset the position of the directory stream to which *dirp* refers to the
62964 beginning of the directory. It shall also cause the directory stream to refer to the current state of
62965 the corresponding directory, as a call to *opendir*( ) would have done. If *dirp* does not refer to a
62966 directory stream, the effect is undefined.

62967 After a call to the *fork*( ) function, either the parent or child (but not both) may continue
62968 XSI processing the directory stream using *readdir*( ), *rewinddir*( ), or *seekdir*( ). If both the parent and
62969 child processes use these functions, the result is undefined.

62970 **RETURN VALUE**

62971 The *rewinddir*( ) function shall not return a value.

62972 **ERRORS**

62973 No errors are defined.

62974 **EXAMPLES**

62975 None.

62976 **APPLICATION USAGE**

62977 The *rewinddir*( ) function should be used in conjunction with *opendir*( ), *readdir*( ), and *closedir*( ) to
62978 examine the contents of the directory. This method is recommended for portability.

62979 **RATIONALE**

62980 None.

62981 **FUTURE DIRECTIONS**

62982 None.

62983 **SEE ALSO**

62984 *closedir*( ), *fdopendir*( ), *readdir*( )

62985 XBD **<dirent.h>**, **<sys/types.h>**

62986 **CHANGE HISTORY**

62987 First released in Issue 2.

62988 **Issue 6**

62989 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

62990 The following new requirements on POSIX implementations derive from alignment with the
62991 Single UNIX Specification:

62992 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
62993 required for conforming implementations of previous POSIX specifications, it was not
62994 required for UNIX applications.

NAME

rint, rintf, rintl — round-to-nearest integral value

SYNOPSIS

```
#include <math.h>

double rint(double x);
float rintf(float x);
long double rintl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall return the integral value (represented as a **double**) nearest *x* in the direction of the current rounding mode. The current rounding mode is implementation-defined.

If the current rounding mode rounds toward negative infinity, then *rint*( ) shall be equivalent to *floor*( ). If the current rounding mode rounds toward positive infinity, then *rint*( ) shall be equivalent to *ceil*( ). If the current rounding mode rounds towards zero, then *rint*( ) shall be MX equivalent to *trunc*( ). If the current rounding mode rounds towards nearest, then *rint*( ) differs from *round*( ) in that halfway cases are rounded to even rather than away from zero.

These functions differ from the *nearbyint*( ), *nearbyintf*( ), and *nearbyintl*( ) functions only in that they may raise the inexact floating-point exception if the result differs in value from the argument.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the integer (represented as a double MX precision number) nearest *x* in the direction of the current rounding mode. The result shall have the same sign as *x*.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The integral value returned by these functions need not be expressible as an **intmax_t**. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

RATIONALE

None.

FUTURE DIRECTIONS

None.

63038 **SEE ALSO**

63039 *abs*( ), *ceil*( ), *feclearexcept*( ), *fetestexcept*( ), *floor*( ), *isnan*( ), *nearbyint*( )

63040 XBD Section 4.23 (on page 109), **<math.h>**

63041 **CHANGE HISTORY**

63042 First released in Issue 4, Version 2.

63043 **Issue 5**

63044 Moved from X/OPEN UNIX extension to BASE.

63045 **Issue 6**

63046 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

63047 • The *rintf*( ) and *rintl*( ) functions are added.

63048 • The *rint*( ) function is no longer marked as an extension.

63049 • The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
63050 revised to align with the ISO/IEC 9899: 1999 standard.

63051 IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard
63052 are marked.

63053 **Issue 7**

63054 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0514 [346], XSH/TC1-2008/0515 [346],
63055 XSH/TC1-2008/0516 [346], XSH/TC1-2008/0517 [346], and XSH/TC1-2008/0518 [346] are
63056 applied.

63057 **NAME**

63058 rmdir — remove a directory

63059 **SYNOPSIS**

63060 `#include <unistd.h>`

63061 `int rmdir(const char *path);`

63062 **DESCRIPTION**

63063 The *rmdir*( ) function shall remove a directory whose name is given by *path*. The directory shall
63064 be removed only if it is an empty directory.

63065 If the directory is the root directory or the current working directory of any process, it is
63066 unspecified whether the function succeeds, or whether it shall fail and set *errno* to [EBUSY].

63067 If *path* names a symbolic link, then *rmdir*( ) shall fail and set *errno* to [ENOTDIR].

63068 If the *path* argument refers to a path whose final component is either dot or dot-dot, *rmdir*( ) shall
63069 fail.

63070 If the directory's link count becomes 0 and no process has the directory open, the space occupied
63071 by the directory shall be freed and the directory shall no longer be accessible. If one or more
63072 processes have the directory open when the last link is removed, the dot and dot-dot entries, if
63073 present, shall be removed before *rmdir*( ) returns and no new entries may be created in the
63074 directory, but the directory shall not be removed until all references to the directory are closed.

63075 If the directory is not an empty directory, *rmdir*( ) shall fail and set *errno* to [EEXIST] or
63076 [ENOTEMPTY].

63077 Upon successful completion, *rmdir*( ) shall mark for update the last data modification and last
63078 file status change timestamps of the parent directory.

63079 **RETURN VALUE**

63080 Upon successful completion, the function *rmdir*( ) shall return 0. Otherwise, –1 shall be returned,
63081 and *errno* set to indicate the error. If –1 is returned, the named directory shall not be changed.

63082 **ERRORS**

63083 The *rmdir*( ) function shall fail if:

63084 [EACCES] Search permission is denied on a component of the path prefix, or write
63085 permission is denied on the parent directory of the directory to be removed.

63086 [EBUSY] The directory to be removed is currently in use by the system or some process
63087 and the implementation considers this to be an error.

63088 [EEXIST] or [ENOTEMPTY]
63089 The *path* argument names a directory that is not an empty directory, or there
63090 are hard links to the directory other than dot or a single entry in dot-dot.

63091 [EINVAL] The *path* argument contains a last component that is dot.

63092 [EIO] A physical I/O error has occurred.

63093 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
63094 argument.

63095 [ENAMETOOLONG]
63096 The length of a component of a pathname is longer than {NAME_MAX}.

63097 [ENOENT] A component of *path* does not name an existing file, or the *path* argument
63098 names a nonexistent directory or points to an empty string.

[ENOTDIR] A component of *path* names an existing file that is neither a directory nor a symbolic link to a directory.

XSI [EPERM] or [EACCES]
The S_ISVTX flag is set on the directory containing the file referred to by the *path* argument and the process does not satisfy the criteria specified in XBD Section 4.5 (on page 96).

[EROFS] The directory entry to be removed resides on a read-only file system.

The *rmdir*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

EXAMPLES

Removing a Directory

The following example shows how to remove a directory named **/home/cnd/mod1**.

```
#include <unistd.h>

int status;
...
status = rmdir("/home/cnd/mod1");
```

APPLICATION USAGE

None.

RATIONALE

The *rmdir*( ) and *rename*( ) functions originated in 4.2 BSD, and they used [ENOTEMPTY] for the condition when the directory to be removed does not exist or *new* already exists. When the 1984 /usr/group standard was published, it contained [EEXIST] instead. When these functions were adopted into System V, the 1984 /usr/group standard was used as a reference. Therefore, several existing applications and implementations support/use both forms, and no agreement could be reached on either value. All implementations are required to supply both [EEXIST] and [ENOTEMPTY] in **<errno.h>** with distinct values, so that applications can use both values in C-language **case** statements.

The meaning of deleting *pathname***/dot** is unclear, because the name of the file (directory) in the parent directory to be removed is not clear, particularly in the presence of multiple links to a directory.

The POSIX.1-1990 standard was silent with regard to the behavior of *rmdir*( ) when there are multiple hard links to the directory being removed. The requirement to set *errno* to [EEXIST] or [ENOTEMPTY] clarifies the behavior in this case.

If the current working directory of the process is being removed, that should be an allowed error.

Virtually all existing implementations detect [ENOTEMPTY] or the case of dot-dot. The text in Section 2.3 (on page 507) about returning any one of the possible errors permits that behavior to continue. The [ELOOP] error may be returned if more than {SYMLOOP_MAX} symbolic links

are encountered during resolution of the *path* argument.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.3 (on page 507), *mkdir( )*, *remove( )*, *rename( )*, *unlink( )*

XBD Section 4.5 (on page 96), **<unistd.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to indicate the results of naming a symbolic link in *path*.
- The [EIO] mandatory error condition is added.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

Austin Group Interpretation 1003.1-2001 #181 is applied, updating the requirements for operations when the S_ISVTX bit is set.

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0519 [324] is applied.

NAME

round, roundf, roundl — round to the nearest integer value in a floating-point format

SYNOPSIS

```
#include <math.h>

double round(double x);
float roundf(float x);
long double roundl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value in floating-point format, rounding halfway cases away from zero, regardless of the current rounding direction.

MX These functions may raise the inexact floating-point exception for finite non-integer arguments.

RETURN VALUE

MX Upon successful completion, these functions shall return the rounded integer value. The result shall have the same sign as *x*.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The integral value returned by these functions need not be expressible as an **intmax_t**. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0520 [346] is applied.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

63206 **NAME**

63207 scalbln, scalblnf, scalblnl, scalbn, scalbnf, scalbnl — compute exponent using FLT_RADIX

63208 **SYNOPSIS**

63209
```
#include <math.h>
```

63210
```
double scalbln(double x, long n);
```
63211
```
float scalblnf(float x, long n);
```
63212
```
long double scalblnl(long double x, long n);
```
63213
```
double scalbn(double x, int n);
```
63214
```
float scalbnf(float x, int n);
```
63215
```
long double scalbnl(long double x, int n);
```

63216 **DESCRIPTION**

63217 CX The functionality described on this reference page is aligned with the ISO C standard. Any
63218 conflict between the requirements described here and the ISO C standard is unintentional. This
63219 volume of POSIX.1-2024 defers to the ISO C standard.

63220 These functions shall compute $x * \text{FLT\_RADIX}^n$ efficiently, not normally by computing
63221 FLT_RADIX^n explicitly.

63222 An application wishing to check for error situations should set *errno* to zero and call
63223 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
63224 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
63225 zero, an error has occurred.

63226 **RETURN VALUE**

63227 Upon successful completion, these functions shall return $x * \text{FLT\_RADIX}^n$.

63228 MX If the calculation does not overflow or underflow, the returned value shall be exact and shall be
63229 independent of the current rounding direction mode.

63230 If the result would cause overflow, a range error shall occur and these functions shall return
63231 ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (according to the sign of *x*) as appropriate for
63232 the return type of the function.

63233 MXX If the correct value would cause underflow, and is not representable, a range error may occur,
63234 MXX and *scalbln*( ), *scalblnf*( ), *scalblnl*( ), *scalbn*( ), *scalbnf*( ), and *scalbnl*( ) shall return 0.0, or (if IEC
63235 60559 Floating-Point is not supported) an implementation-defined value no greater in
63236 magnitude than DBL_MIN, FLT_MIN, LDBL_MIN, DBL_MIN, FLT_MIN, and LDBL_MIN,
63237 respectively.

63238 MX If *x* is NaN, a NaN shall be returned.

63239 If *x* is ±0 or ±Inf, *x* shall be returned.

63240 If *n* is 0, *x* shall be returned.

63241 MXX If the correct value would cause underflow, and is representable, a range error may occur and
63242 the correct value shall be returned.

63243 **ERRORS**

63244 These functions shall fail if:

63245 Range Error The result overflows.

63246 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
63247 then *errno* shall be set to [ERANGE]. If the integer expression
63248 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
63249 floating-point exception shall be raised.

These functions may fail if:

Range Error — The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions are named so as to avoid conflicting with the historical definition of the *scalb*( ) function from the Single UNIX Specification. The difference is that the *scalb*( ) function has a second argument of **double** instead of **int**. The *scalb*( ) function is not part of the ISO C standard. The three functions whose second type is **long** are provided because the factor required to scale from the smallest positive floating-point value to the largest finite one, on many implementations, is too large to represent in the minimum-width **int** format.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0521 [68] and XSH/TC1-2008/0522 [68] are applied.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

63281 **NAME**

63282 scandir — scan a directory

63283 **SYNOPSIS**

63284 `#include <dirent.h>`

63285 `int scandir(const char *dir, struct dirent ***namelist,`
63286 `int (*sel)(const struct dirent *),`
63287 `int (*compar)(const struct dirent **, const struct dirent **));`

63288 **DESCRIPTION**

63289 Refer to *alphasort( )*.

63290 **NAME**

63291 scanf — convert formatted input

63292 **SYNOPSIS**

63293 `#include <stdio.h>`

63294 `int scanf(const char *restrict format, ...);`

63295 **DESCRIPTION**

63296 Refer to *fscanf( )*.

63297 **NAME**

63298 sched_get_priority_max, sched_get_priority_min — get priority limits (**REALTIME**)

63299 **SYNOPSIS**

63300 PS|TPS
```
#include <sched.h>

int sched_get_priority_max(int policy);
int sched_get_priority_min(int policy);
```

63303 **DESCRIPTION**

63304 The *sched_get_priority_max*( ) and *sched_get_priority_min*( ) functions shall return the appropriate
63305 maximum or minimum, respectively, for the scheduling policy specified by *policy*.

63306 The value of *policy* shall be one of the scheduling policy values defined in **<sched.h>**.

63307 **RETURN VALUE**

63308 If successful, the *sched_get_priority_max*( ) and *sched_get_priority_min*( ) functions shall return the
63309 appropriate maximum or minimum values, respectively. If unsuccessful, they shall return a
63310 value of –1 and set *errno* to indicate the error.

63311 **ERRORS**

63312 The *sched_get_priority_max*( ) and *sched_get_priority_min*( ) functions shall fail if:

63313 [EINVAL] The value of the *policy* parameter does not represent a defined scheduling
63314 policy.

63315 **EXAMPLES**

63316 None.

63317 **APPLICATION USAGE**

63318 None.

63319 **RATIONALE**

63320 None.

63321 **FUTURE DIRECTIONS**

63322 None.

63323 **SEE ALSO**

63324 *sched_getparam*( ), *sched_setparam*( ), *sched_getscheduler*( ), *sched_rr_get_interval*( ),
63325 *sched_setscheduler*( )

63326 XBD **<sched.h>**

63327 **CHANGE HISTORY**

63328 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

63329 **Issue 6**

63330 These functions are marked as part of the Process Scheduling option.

63331 The [ENOSYS] error condition has been removed as stubs need not be provided if an
63332 implementation does not support the Process Scheduling option.

63333 The [ESRCH] error condition has been removed since these functions do not take a *pid*
63334 argument.

63335 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/52 is applied, changing the PS margin
63336 code in the SYNOPSIS to PS|TPS.

63337 **NAME**

63338 sched_getparam — get scheduling parameters (**REALTIME**)

63339 **SYNOPSIS**

63340 PS
```
#include <sched.h>

int sched_getparam(pid_t pid, struct sched_param *param);
```
63341

63342 **DESCRIPTION**

63343 The *sched_getparam*( ) function shall return the scheduling parameters of a process specified by
63344 *pid* in the **sched_param** structure pointed to by *param*.

63345 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
63346 parameters for the process whose process ID is equal to *pid* shall be returned.

63347 If *pid* is zero, the scheduling parameters for the calling process shall be returned. The behavior of
63348 the *sched_getparam*( ) function is unspecified if the value of *pid* is negative.

63349 **RETURN VALUE**

63350 Upon successful completion, the *sched_getparam*( ) function shall return zero. If the call to
63351 *sched_getparam*( ) is unsuccessful, the function shall return a value of −1 and set *errno* to indicate
63352 the error.

63353 **ERRORS**

63354 The *sched_getparam*( ) function shall fail if:

63355 [EPERM] The requesting process does not have permission to obtain the scheduling
63356 parameters of the specified process.

63357 [ESRCH] No process can be found corresponding to that specified by *pid*.

63358 **EXAMPLES**

63359 None.

63360 **APPLICATION USAGE**

63361 None.

63362 **RATIONALE**

63363 None.

63364 **FUTURE DIRECTIONS**

63365 None.

63366 **SEE ALSO**

63367 *sched_getscheduler*( ), *sched_setparam*( ), *sched_setscheduler*( )

63368 XBD **<sched.h>**

63369 **CHANGE HISTORY**

63370 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

63371 **Issue 6**

63372 The *sched_getparam*( ) function is marked as part of the Process Scheduling option.

63373 The [ENOSYS] error condition has been removed as stubs need not be provided if an
63374 implementation does not support the Process Scheduling option.

63375 **NAME**

63376 sched_getscheduler — get scheduling policy (**REALTIME**)

63377 **SYNOPSIS**

63378 PS
```
#include <sched.h>

int sched_getscheduler(pid_t pid);
```

63379

63380 **DESCRIPTION**

63381 The *sched_getscheduler*( ) function shall return the scheduling policy of the process specified by
63382 *pid*. If the value of *pid* is negative, the behavior of the *sched_getscheduler*( ) function is
63383 unspecified.

63384 The values that can be returned by *sched_getscheduler*( ) are defined in the **<sched.h>** header.

63385 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
63386 policy shall be returned for the process whose process ID is equal to *pid*.

63387 If *pid* is zero, the scheduling policy shall be returned for the calling process.

63388 **RETURN VALUE**

63389 Upon successful completion, the *sched_getscheduler*( ) function shall return the scheduling policy
63390 of the specified process. If unsuccessful, the function shall return –1 and set *errno* to indicate the
63391 error.

63392 **ERRORS**

63393 The *sched_getscheduler*( ) function shall fail if:

63394 [EPERM] The requesting process does not have permission to determine the scheduling
63395 policy of the specified process.

63396 [ESRCH] No process can be found corresponding to that specified by *pid*.

63397 **EXAMPLES**

63398 None.

63399 **APPLICATION USAGE**

63400 None.

63401 **RATIONALE**

63402 None.

63403 **FUTURE DIRECTIONS**

63404 None.

63405 **SEE ALSO**

63406 *sched_getparam*( ), *sched_setparam*( ), *sched_setscheduler*( )

63407 XBD **<sched.h>**

63408 **CHANGE HISTORY**

63409 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

63410 **Issue 6**

63411 The *sched_getscheduler*( ) function is marked as part of the Process Scheduling option.

63412 The [ENOSYS] error condition has been removed as stubs need not be provided if an
63413 implementation does not support the Process Scheduling option.

NAME

sched_rr_get_interval — get execution time limits (**REALTIME**)

SYNOPSIS

PS|TPS

```
#include <sched.h>

int sched_rr_get_interval(pid_t pid, struct timespec *interval);
```

DESCRIPTION

The *sched_rr_get_interval*( ) function shall update the **timespec** structure referenced by the *interval* argument to contain the current execution time limit (that is, time quantum) for the process specified by *pid*. If *pid* is zero, the current execution time limit for the calling process shall be returned.

RETURN VALUE

If successful, the *sched_rr_get_interval*( ) function shall return zero. Otherwise, it shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *sched_rr_get_interval*( ) function shall fail if:

[ESRCH] No process can be found corresponding to that specified by *pid*.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sched_getparam( ), *sched_get_priority_max*( ), *sched_getscheduler*( ), *sched_setparam*( ), *sched_setscheduler*( )

XBD **<sched.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *sched_rr_get_interval*( ) function is marked as part of the Process Scheduling option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Scheduling option.

IEEE Std 1003.1-2001/Cor 1-2002, XSH/TC1/D6/53 is applied, changing the PS margin code in the SYNOPSIS to PS|TPS.

NAME

sched_setparam — set scheduling parameters (**REALTIME**)

SYNOPSIS

PS
```
#include <sched.h>

int sched_setparam(pid_t pid, const struct sched_param *param);
```

DESCRIPTION

The *sched_setparam*( ) function shall set the scheduling parameters of the process specified by *pid* to the values specified by the **sched_param** structure pointed to by *param*. The value of the *sched_priority* member in the **sched_param** structure shall be any integer within the inclusive priority range for the current scheduling policy of the process specified by *pid*. Higher numerical values for the priority represent higher priorities. If the value of *pid* is negative, the behavior of the *sched_setparam*( ) function is unspecified.

If a process specified by *pid* exists, and if the calling process has permission, the scheduling parameters shall be set for the process whose process ID is equal to *pid*.

If *pid* is zero, the scheduling parameters shall be set for the calling process.

The conditions under which one process has permission to change the scheduling parameters of another process are implementation-defined.

Implementations may require the requesting process to have appropriate privileges to set its own scheduling parameters or those of another process.

See Scheduling Policies (on page 531) for a description on how this function affects the scheduling of the threads within the target process.

SS If the current scheduling policy for the target process is not SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC, the result is implementation-defined; this case includes the SCHED_OTHER policy.

SS The specified *sched_ss_repl_period* shall be greater than or equal to the specified *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.

The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the function to succeed; if not, the function shall fail. It is unspecified whether the *sched_ss_repl_period* and *sched_ss_init_budget* values are stored as provided by this function or are rounded to align with the resolution of the clock being used.

This function is not atomic with respect to other threads in the process. Threads may continue to execute while this function call is in the process of changing the scheduling policy for the underlying kernel-scheduled entities used by the process contention scope threads.

RETURN VALUE

If successful, the *sched_setparam*( ) function shall return zero.

If the call to *sched_setparam*( ) is unsuccessful, the priority shall remain unchanged, and the function shall return a value of –1 and set *errno* to indicate the error.

ERRORS

The *sched_setparam*( ) function shall fail if:

[EINVAL] One or more of the requested scheduling parameters is outside the range defined for the scheduling policy of the specified *pid*.

63491 [EPERM] The requesting process does not have permission to set the scheduling
63492 parameters for the specified process, or does not have appropriate privileges
63493 to invoke *sched_setparam*( ).

63494 [ESRCH] No process can be found corresponding to that specified by *pid*.

63495 **EXAMPLES**

63496 None.

63497 **APPLICATION USAGE**

63498 None.

63499 **RATIONALE**

63500 None.

63501 **FUTURE DIRECTIONS**

63502 None.

63503 **SEE ALSO**

63504 Scheduling Policies (on page 531), *sched_getparam*( ), *sched_getscheduler*( ), *sched_setscheduler*( )

63505 XBD **<sched.h>**

63506 **CHANGE HISTORY**

63507 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

63508 **Issue 6**

63509 The *sched_setparam*( ) function is marked as part of the Process Scheduling option.

63510 The [ENOSYS] error condition has been removed as stubs need not be provided if an
63511 implementation does not support the Process Scheduling option.

63512 The following new requirements on POSIX implementations derive from alignment with the
63513 Single UNIX Specification:

63514 • In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is
63515 added.

63516 • Sections describing two-level scheduling and atomicity of the function are added.

63517 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

63518 IEEE PASC Interpretation 1003.1 #100 is applied.

63519 **Issue 7**

63520 Austin Group Interpretation 1003.1-2001 #061 is applied, updating the DESCRIPTION.

63521 Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements
63522 for the *sched_ss_repl_period* and *sched_ss_init_budget* values.

NAME

sched_setscheduler — set scheduling policy and parameters (**REALTIME**)

SYNOPSIS

PS
```
#include <sched.h>

int sched_setscheduler(pid_t pid, int policy,
    const struct sched_param *param);
```

DESCRIPTION

The *sched_setscheduler*( ) function shall set the scheduling policy and scheduling parameters of the process specified by *pid* to *policy* and the parameters specified in the **sched_param** structure pointed to by *param*, respectively. The value of the *sched_priority* member in the **sched_param** structure shall be any integer within the inclusive priority range for the scheduling policy specified by *policy*. If the value of *pid* is negative, the behavior of the *sched_setscheduler*( ) function is unspecified.

The possible values for the *policy* parameter are defined in the **<sched.h>** header.

If a process specified by *pid* exists, and if the calling process has permission, the scheduling policy and scheduling parameters shall be set for the process whose process ID is equal to *pid*.

If *pid* is zero, the scheduling policy and scheduling parameters shall be set for the calling process.

The conditions under which one process has appropriate privileges to change the scheduling parameters of another process are implementation-defined.

Implementations may require that the requesting process have permission to set its own scheduling parameters or those of another process. Additionally, implementation-defined restrictions may apply as to the appropriate privileges required to set the scheduling policy of the process, or the scheduling policy of another process, to a particular value.

The *sched_setscheduler*( ) function shall be considered successful if it succeeds in setting the scheduling policy and scheduling parameters of the process specified by *pid* to the values specified by *policy* and the structure pointed to by *param*, respectively.

See Scheduling Policies (on page 531) for a description on how this function affects the scheduling of the threads within the target process.

SS If the current scheduling policy for the target process is not SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC, the result is implementation-defined; this case includes the SCHED_OTHER policy.

SS The specified *sched_ss_repl_period* shall be greater than or equal to the specified *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.

The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the function to succeed; if not, the function shall fail. It is unspecified whether the *sched_ss_repl_period* and *sched_ss_init_budget* values are stored as provided by this function or are rounded to align with the resolution of the clock being used.

This function is not atomic with respect to other threads in the process. Threads may continue to execute while this function call is in the process of changing the scheduling policy and associated scheduling parameters for the underlying kernel-scheduled entities used by the process contention scope threads.

63565 **RETURN VALUE**

63566 Upon successful completion, the function shall return the former scheduling policy of the
63567 specified process. If the *sched_setscheduler*( ) function fails to complete successfully, the policy
63568 and scheduling parameters shall remain unchanged, and the function shall return a value of −1
63569 and set *errno* to indicate the error.

63570 **ERRORS**

63571 The *sched_setscheduler*( ) function shall fail if:

63572 [EINVAL] The value of the *policy* parameter is invalid, or one or more of the parameters
63573 contained in *param* is outside the valid range for the specified scheduling
63574 policy.

63575 [EPERM] The requesting process does not have permission to set either or both of the
63576 scheduling parameters or the scheduling policy of the specified process.

63577 [ESRCH] No process can be found corresponding to that specified by *pid*.

63578 **EXAMPLES**

63579 None.

63580 **APPLICATION USAGE**

63581 None.

63582 **RATIONALE**

63583 None.

63584 **FUTURE DIRECTIONS**

63585 None.

63586 **SEE ALSO**

63587 Scheduling Policies (on page 531), *sched_getparam*( ), *sched_getscheduler*( ), *sched_setparam*( )

63588 XBD **<sched.h>**

63589 **CHANGE HISTORY**

63590 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

63591 **Issue 6**

63592 The *sched_setscheduler*( ) function is marked as part of the Process Scheduling option.

63593 The [ENOSYS] error condition has been removed as stubs need not be provided if an
63594 implementation does not support the Process Scheduling option.

63595 The following new requirements on POSIX implementations derive from alignment with the
63596 Single UNIX Specification:

63597 • In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is
63598 added.

63599 • Sections describing two-level scheduling and atomicity of the function are added.

63600 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

63601 **Issue 7**

63602 Austin Group Interpretation 1003.1-2001 #061 is applied, updating the DESCRIPTION.

63603 Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements
63604 for the *sched_ss_repl_period* and *sched_ss_init_budget* values.

63605 **NAME**

63606 sched_yield — yield the processor

63607 **SYNOPSIS**

63608
```
#include <sched.h>
```

63609
```
int sched_yield(void);
```

63610 **DESCRIPTION**

63611 The *sched_yield*( ) function shall force the running thread to relinquish the processor until it again
63612 becomes the head of its thread list. It takes no arguments.

63613 **RETURN VALUE**

63614 The *sched_yield*( ) function shall return 0 if it completes successfully; otherwise, it shall return a
63615 value of –1 and set *errno* to indicate the error.

63616 **ERRORS**

63617 No errors are defined.

63618 **EXAMPLES**

63619 None.

63620 **APPLICATION USAGE**

63621 The conceptual model for scheduling semantics in POSIX.1-2024 defines a set of thread lists. This
63622 set of thread lists is always present regardless of the scheduling options supported by the
63623 system. On a system where the Process Scheduling option is not supported, portable
63624 applications should not make any assumptions regarding whether threads from other processes
63625 will be on the same thread list.

63626 **RATIONALE**

63627 None.

63628 **FUTURE DIRECTIONS**

63629 None.

63630 **SEE ALSO**

63631 XBD **<sched.h>**

63632 **CHANGE HISTORY**

63633 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the
63634 POSIX Threads Extension.

63635 **Issue 6**

63636 The *sched_yield*( ) function is now marked as part of the Process Scheduling and Threads options.

63637 **Issue 7**

63638 SD5-XSH-ERN-120 is applied, adding APPLICATION USAGE.

63639 The *sched_yield*( ) function is moved to the Base.

63640 **NAME**

63641 secure_getenv — get value of an environment variable

63642 **SYNOPSIS**

63643 ```
#include <stdlib.h>
```

63644 CX ```
char *secure_getenv(const char *name);
```

63645 **DESCRIPTION**

63646 Refer to *getenv( )*.

NAME

seed48 — seed a uniformly distributed pseudo-random non-negative long integer generator

SYNOPSIS

XSI
```
#include <stdlib.h>

unsigned short *seed48(unsigned short seed16v[3]);
```

DESCRIPTION

Refer to *drand48( )*.

63654 **NAME**

63655 seekdir — set the position of a directory stream

63656 **SYNOPSIS**

63657 XSI
```
#include <dirent.h>

void seekdir(DIR *dirp, long loc);
```

63659 **DESCRIPTION**

63660 The *seekdir*( ) function shall set the position of the next *readdir*( ) operation on the directory
63661 stream specified by *dirp* to the position specified by *loc*. The value of *loc* should have been
63662 returned from an earlier call to *telldir*( ) using the same directory stream. The new position
63663 reverts to the one associated with the directory stream when *telldir*( ) was performed.

63664 If the value of *loc* was not obtained from an earlier call to *telldir*( ), or if a call to *rewinddir*( )
63665 occurred between the call to *telldir*( ) and the call to *seekdir*( ), the results of subsequent calls to
63666 *readdir*( ) are unspecified.

63667 **RETURN VALUE**

63668 The *seekdir*( ) function shall not return a value.

63669 **ERRORS**

63670 No errors are defined.

63671 **EXAMPLES**

63672 None.

63673 **APPLICATION USAGE**

63674 None.

63675 **RATIONALE**

63676 The original standard developers perceived that there were restrictions on the use of the
63677 *seekdir*( ) and *telldir*( ) functions related to implementation details, and for that reason these
63678 functions need not be supported on all POSIX-conforming systems. They are required on
63679 implementations supporting the XSI option.

63680 One of the perceived problems of implementation is that returning to a given point in a directory
63681 is quite difficult to describe formally, in spite of its intuitive appeal, when systems that use B-
63682 trees, hashing functions, or other similar mechanisms to order their directories are considered.
63683 The definition of *seekdir*( ) and *telldir*( ) does not specify whether, when using these interfaces, a
63684 given directory entry will be seen at all, or more than once.

63685 On systems not supporting these functions, their capability can sometimes be accomplished by
63686 saving a filename found by *readdir*( ) and later using *rewinddir*( ) and a loop on *readdir*( ) to
63687 relocate the position from which the filename was saved.

63688 **FUTURE DIRECTIONS**

63689 None.

63690 **SEE ALSO**

63691 *fdopendir*( ), *readdir*( ), *telldir*( )

63692 XBD **<dirent.h>**, **<sys/types.h>**

63693 **CHANGE HISTORY**

63694 First released in Issue 2.

Issue 6

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

Issue 7

SD5-XSH-ERN-200 is applied, updating the DESCRIPTION to note that the value of *loc* should have been returned from an earlier call to *telldir*( ) using the same directory stream.

63700 **NAME**

63701 select — synchronous I/O multiplexing

63702 **SYNOPSIS**

63703 ```
#include <sys/select.h>
```

63704 ```
int select(int nfds, fd_set *restrict readfds,
```

63705 ```
    fd_set *restrict writefds, fd_set *restrict errorfds,
```

63706 ```
    struct timeval *restrict timeout);
```

63707 **DESCRIPTION**

63708 Refer to *pselect( )*.

63709 **NAME**

63710 sem_clockwait, sem_timedwait — lock a semaphore

63711 **SYNOPSIS**

63712
```
#include <semaphore.h>
```

63713
```
int sem_clockwait(sem_t *restrict sem, clockid_t clock_id,
    const struct timespec *restrict abstime);
```
63714
63715
```
int sem_timedwait(sem_t *restrict sem,
```
63716
```
    const struct timespec *restrict abstime);
```

63717 **DESCRIPTION**

63718 The *sem_clockwait*( ) and *sem_timedwait*( ) functions shall lock the semaphore referenced by *sem* as
63719 in the *sem_wait*( ) function. However, if the semaphore cannot be locked without waiting for
63720 another process or thread to unlock the semaphore by performing a *sem_post*( ) function, this
63721 wait shall be terminated when the specified timeout expires.

63722 The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the
63723 clock on which timeouts are based (that is, when the value of that clock equals or exceeds
63724 *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the
63725 call.

63726 For *sem_timedwait*( ), the timeout shall be based on the CLOCK_REALTIME clock. For
63727 *sem_clockwait*( ), the timeout shall be based on the clock specified by the *clock_id* argument. The
63728 resolution of the timeout shall be the resolution of the clock on which it is based.
63729 Implementations shall support passing CLOCK_REALTIME and CLOCK_MONOTONIC to
63730 *sem_clockwait*( ) as the *clock_id* argument.

63731 Under no circumstance shall the function fail with a timeout if the semaphore can be locked
63732 immediately. The validity of the *abstime* need not be checked if the semaphore can be locked
63733 immediately.

63734 **RETURN VALUE**

63735 The *sem_clockwait*( ) and *sem_timedwait*( ) functions shall return zero if the calling process
63736 successfully performed the semaphore lock operation on the semaphore designated by *sem*. If
63737 the call was unsuccessful, the state of the semaphore shall be unchanged, and the functions shall
63738 return a value of –1 and set *errno* to indicate the error.

63739 **ERRORS**

63740 The *sem_clockwait*( ) and *sem_timedwait*( ) functions shall fail if:

63741 [EINVAL] The process or thread would have blocked, and either the *abstime* parameter
63742 specified a nanoseconds field value less than zero or greater than or equal to
63743 1 000 million, or the *sem_clockwait*( ) function was passed an invalid or
63744 unsupported *clock_id* value.

63745 [ETIMEDOUT] The semaphore could not be locked before the specified timeout expired.

63746 The *sem_clockwait*( ) and *sem_timedwait*( ) functions may fail if:

63747 [EDEADLK] A deadlock condition was detected.

63748 [EINTR] A signal interrupted the function.

63749 [EINVAL] The *sem* argument does not refer to a valid semaphore.

EXAMPLES

The program shown below operates on an unnamed semaphore. The program expects two command-line arguments. The first argument specifies a seconds value that is used to set an alarm timer to generate a SIGALRM signal. This handler performs a *sem_post*( ) to increment the semaphore that is being waited on in *main( )* using *sem_clockwait*( ). The second command-line argument specifies the length of the timeout, in seconds, for *sem_clockwait*( ).

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <time.h>
#include <assert.h>
#include <errno.h>
#include <signal.h>

sem_t sem;

static void
handler(int sig)
{
    int sav_errno = errno;
    static const char info_msg[] = "sem_post() from handler\n";
    write(STDOUT_FILENO, info_msg, sizeof info_msg - 1);
    if (sem_post(&sem) == -1) {
        static const char err_msg[] = "sem_post() failed\n";
        write(STDERR_FILENO, err_msg, sizeof err_msg - 1);
        _exit(EXIT_FAILURE);
    }
    errno = sav_errno;
}

int
main(int argc, char *argv[])
{
    struct sigaction sa;
    struct timespec ts;
    int s;

    if (argc != 3) {
        fprintf(stderr, "Usage: %s <alarm-secs> <wait-secs>\n",
            argv[0]);
        exit(EXIT_FAILURE);
    }

    if (sem_init(&sem, 0, 0) == -1) {
        perror("sem_init");
        exit(EXIT_FAILURE);
    }

    /* Establish SIGALRM handler; set alarm timer using argv[1] */

    sa.sa_handler = handler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    if (sigaction(SIGALRM, &sa, NULL) == -1) {
```



```
63798                perror("sigaction");
63799                exit(EXIT_FAILURE);
63800            }
63801            alarm(atoi(argv[1]));
63802            /* Calculate relative interval as current time plus
63803               number of seconds given argv[2] */
63804            if (clock_gettime(CLOCK_MONOTONIC, &ts) == -1) {
63805                perror("clock_gettime");
63806                exit(EXIT_FAILURE);
63807            }
63808            ts.tv_sec += atoi(argv[2]);
63809            printf("main() about to call sem_clockwait()\n");
63810            while ((s = sem_clockwait(&sem, CLOCK_MONOTONIC, &ts)) == -1 &&
63811                  errno == EINTR)
63812                continue;       /* Restart if interrupted by handler */
63813            /* Check what happened */
63814            if (s == -1) {
63815                if (errno == ETIMEDOUT)
63816                    printf("sem_clockwait() timed out\n");
63817                else
63818                    perror("sem_clockwait");
63819            } else
63820                printf("sem_clockwait() succeeded\n");
63821            exit((s == 0) ? EXIT_SUCCESS : EXIT_FAILURE);
63822        }
```

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in XBD Section 3.275 (on page 72).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sem_post( ), *sem_trywait( )*, *semctl( )*, *semget( )*, *semop( )*, *time( )*

XBD Section 3.275 (on page 72), **<semaphore.h>**, **<time.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/120 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

Issue 7

The *sem_timedwait( )* function is moved from the Semaphores option to the Base.

Functionality relating to the Timers option is moved to the Base.

An example is added.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0529 [138] is applied.

Issue 8

Austin Group Defect 592 is applied, removing text relating to **<time.h>** from the SYNOPSIS and DESCRIPTION sections.

Austin Group Defect 1216 is applied, adding *sem_clockwait*( ).

63846 **NAME**

63847 sem_close — close a named semaphore

63848 **SYNOPSIS**

63849 `#include <semaphore.h>`

63850 `int sem_close(sem_t *sem);`

63851 **DESCRIPTION**

63852 The *sem_close*( ) function shall indicate that the calling process is finished using the named
63853 semaphore indicated by *sem*. The effects of calling *sem_close*( ) for an unnamed semaphore (one
63854 created by *sem_init*( )) are undefined. The *sem_close*( ) function shall deallocate (that is, make
63855 available for reuse by a subsequent *sem_open*( ) by this process) any system resources allocated
63856 by the system for use by this process for this semaphore. If the semaphore indicated by *sem* is
63857 implemented using a file descriptor, the file descriptor shall be closed. The effect of subsequent
63858 use of the semaphore indicated by *sem* by this process is undefined. If any threads in the calling
63859 process are currently blocked on the semaphore, the behavior is undefined. If the semaphore
63860 has not been removed with a successful call to *sem_unlink*( ), then *sem_close*( ) has no effect on the
63861 state of the semaphore. If the *sem_unlink*( ) function has been successfully invoked for *name* after
63862 the most recent call to *sem_open*( ) with O_CREAT for this semaphore, then when all processes
63863 that have opened the semaphore close all semaphore handles to it, the semaphore is no longer
63864 accessible.

63865 **RETURN VALUE**

63866 Upon successful completion, a value of zero shall be returned. Otherwise, a value of –1 shall be
63867 returned and *errno* set to indicate the error.

63868 **ERRORS**

63869 The *sem_close*( ) function may fail if:

63870 [EINVAL] The *sem* argument is not a valid semaphore descriptor.

63871 **EXAMPLES**

63872 None.

63873 **APPLICATION USAGE**

63874 None.

63875 **RATIONALE**

63876 None.

63877 **FUTURE DIRECTIONS**

63878 None.

63879 **SEE ALSO**

63880 *semctl*( ), *semget*( ), *semop*( ), *sem_init*( ), *sem_open*( ), *sem_unlink*( )

63881 XBD **<semaphore.h>**

63882 **CHANGE HISTORY**

63883 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

63884 **Issue 6**

63885 The *sem_close*( ) function is marked as part of the Semaphores option.

63886 The [ENOSYS] error condition has been removed as stubs need not be provided if an
63887 implementation does not support the Semaphores option.

63888 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/113 is applied, updating the ERRORS
63889 section so that the [EINVAL] error becomes optional.

Issue 7

The *sem_close*( ) function is moved from the Semaphores option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0523 [37] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0317 [870] is applied.

Issue 8

Austin Group Defect 368 is applied, adding a requirement that if *sem* is implemented using a file descriptor, *sem_close*( ) closes the file descriptor.

Austin Group Defect 1324 is applied, clarifying the circumstances under which an unlinked semaphore is no longer accessible.

63899 **NAME**

63900 sem_destroy — destroy an unnamed semaphore

63901 **SYNOPSIS**

63902 `#include <semaphore.h>`

63903 `int sem_destroy(sem_t *sem);`

63904 **DESCRIPTION**

63905 The *sem_destroy*( ) function shall destroy the unnamed semaphore indicated by *sem*. If an
63906 unnamed semaphore is implemented using a file descriptor, the file descriptor shall be closed.
63907 Only a semaphore that was created using *sem_init*( ) can be destroyed using *sem_destroy*( ); the
63908 effect of calling *sem_destroy*( ) with a named semaphore is undefined. The effect of subsequent
63909 use of the semaphore *sem* is undefined until *sem* is reinitialized by another call to *sem_init*( ).

63910 It is safe to destroy an initialized semaphore upon which no threads are currently blocked. The
63911 effect of destroying a semaphore upon which other threads are currently blocked is undefined.

63912 **RETURN VALUE**

63913 Upon successful completion, a value of zero shall be returned. Otherwise, a value of –1 shall be
63914 returned and *errno* set to indicate the error.

63915 **ERRORS**

63916 The *sem_destroy*( ) function may fail if:

63917 [EINVAL] The *sem* argument is not a valid semaphore.

63918 [EBUSY] There are currently processes blocked on the semaphore.

63919 **EXAMPLES**

63920 None.

63921 **APPLICATION USAGE**

63922 None.

63923 **RATIONALE**

63924 None.

63925 **FUTURE DIRECTIONS**

63926 None.

63927 **SEE ALSO**

63928 *semctl*( ), *semget*( ), *semop*( ), *sem_init*( ), *sem_open*( )

63929 XBD **<semaphore.h>**

63930 **CHANGE HISTORY**

63931 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

63932 **Issue 6**

63933 The *sem_destroy*( ) function is marked as part of the Semaphores option.

63934 The [ENOSYS] error condition has been removed as stubs need not be provided if an
63935 implementation does not support the Semaphores option.

63936 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/114 is applied, updating the ERRORS
63937 section so that the [EINVAL] error becomes optional.

Issue 7

The *sem_destroy*( ) function is moved from the Semaphores option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0524 [37] is applied.

Issue 8

Austin Group Defect 368 is applied, adding a requirement that if an unnamed semaphore is implemented using a file descriptor, *sem_destroy*( ) closes the file descriptor.

63944 **NAME**

63945 sem_getvalue — get the value of a semaphore

63946 **SYNOPSIS**

63947 `#include <semaphore.h>`

63948 `int sem_getvalue(sem_t *restrict sem, int *restrict sval);`

63949 **DESCRIPTION**

63950 The *sem_getvalue*( ) function shall update the location referenced by the *sval* argument to have
63951 the value of the semaphore referenced by *sem* without affecting the state of the semaphore. The
63952 updated value represents an actual semaphore value that occurred at some unspecified time
63953 during the call, but it need not be the actual value of the semaphore when it is returned to the
63954 calling process.

63955 If *sem* is locked, then the object to which *sval* points shall either be set to zero or to a negative
63956 number whose absolute value represents the number of processes waiting for the semaphore at
63957 some unspecified time during the call.

63958 **RETURN VALUE**

63959 Upon successful completion, the *sem_getvalue*( ) function shall return a value of zero. Otherwise,
63960 it shall return a value of –1 and set *errno* to indicate the error.

63961 **ERRORS**

63962 The *sem_getvalue*( ) function may fail if:

63963 [EINVAL] The *sem* argument does not refer to a valid semaphore.

63964 **EXAMPLES**

63965 None.

63966 **APPLICATION USAGE**

63967 None.

63968 **RATIONALE**

63969 None.

63970 **FUTURE DIRECTIONS**

63971 None.

63972 **SEE ALSO**

63973 *semctl*( ), *semget*( ), *semop*( ), *sem_clockwait*( ), *sem_post*( ), *sem_trywait*( )

63974 XBD **<semaphore.h>**

63975 **CHANGE HISTORY**

63976 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

63977 **Issue 6**

63978 The *sem_getvalue*( ) function is marked as part of the Semaphores option.

63979 The [ENOSYS] error condition has been removed as stubs need not be provided if an
63980 implementation does not support the Semaphores option.

63981 The *sem_timedwait*( ) function is added to the SEE ALSO section for alignment with IEEE Std
63982 1003.1d-1999.

63983 The **restrict** keyword is added to the *sem_getvalue*( ) prototype for alignment with the
63984 ISO/IEC 9899: 1999 standard.

63985 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/54 is applied.

63986 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/115 is applied, updating the ERRORS

section so that the [EINVAL] error becomes optional.

Issue 7

The *sem_getvalue*( ) function is moved from the Semaphores option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0525 [37] is applied.

63991 **NAME**

63992 sem_init — initialize an unnamed semaphore

63993 **SYNOPSIS**

63994 `#include <semaphore.h>`

63995 `int sem_init(sem_t *sem, int pshared, unsigned value);`

63996 **DESCRIPTION**

63997 The *sem_init*( ) function shall initialize the unnamed semaphore referred to by *sem*. The value of
63998 the initialized semaphore shall be *value*. Following a successful call to *sem_init*( ), the semaphore
63999 can be used in subsequent calls to *sem_clockwait*( ), *sem_destroy*( ), *sem_post*( ), *sem_timedwait*( ),
64000 *sem_trywait*( ), and *sem_wait*( ). This semaphore shall remain usable until the semaphore is
64001 destroyed. An unnamed semaphore may be implemented using a file descriptor.

64002 If the *pshared* argument has a non-zero value, then the semaphore is shared between processes;
64003 in this case, any process that can access the semaphore *sem* can use *sem* for performing
64004 *sem_clockwait*( ), *sem_destroy*( ), *sem_post*( ), *sem_timedwait*( ), *sem_trywait*( ), and *sem_wait*( )
64005 operations.

64006 If the *pshared* argument is zero, then the semaphore is shared between threads of the process; any
64007 thread in this process can use *sem* for performing *sem_clockwait*( ), *sem_destroy*( ), *sem_post*( ),
64008 *sem_timedwait*( ), *sem_trywait*( ), and *sem_wait*( ) operations.

64009 See Section 2.9.9 (on page 548) for further requirements.

64010 Attempting to initialize an already initialized semaphore results in undefined behavior.

64011 **RETURN VALUE**

64012 Upon successful completion, the *sem_init*( ) function shall initialize the semaphore in *sem* and
64013 return 0. Otherwise, it shall return –1 and set *errno* to indicate the error.

64014 **ERRORS**

64015 The *sem_init*( ) function shall fail if:

64016 [EINVAL] The *value* argument exceeds {SEM_VALUE_MAX}.

64017 [ENOSPC] A resource required to initialize the semaphore has been exhausted, or the
64018 limit on semaphores ({SEM_NSEMS_MAX}) has been reached.

64019 [EPERM] The process lacks appropriate privileges to initialize the semaphore.

64020 The *sem_init*( ) function may fail if:

64021 [EMFILE] All file descriptors available to the process are currently open.

64022 [ENFILE] The maximum allowable number of files is currently open in the system.

64023 **EXAMPLES**

64024 None.

64025 **APPLICATION USAGE**

64026 None.

64027 **RATIONALE**

64028 None.

64029 **FUTURE DIRECTIONS**

64030 None.

64031 **SEE ALSO**

64032 *sem_clockwait*( ), *sem_destroy*( ), *sem_post*( ), *sem_trywait*( )

64033 XBD **<semaphore.h>**

64034 **CHANGE HISTORY**

64035 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

64036 **Issue 6**

64037 The *sem_init*( ) function is marked as part of the Semaphores option.

64038 The [ENOSYS] error condition has been removed as stubs need not be provided if an
64039 implementation does not support the Semaphores option.

64040 The *sem_timedwait*( ) function is added to the SEE ALSO section for alignment with IEEE Std
64041 1003.1d-1999.

64042 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/116 is applied, updating the
64043 DESCRIPTION to add the *sem_timedwait*( ) function for alignment with IEEE Std 1003.1d-1999.

64044 **Issue 7**

64045 SD5-XSH-ERN-176 is applied.

64046 The *sem_init*( ) function is moved from the Semaphores option to the Base.

64047 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0526 [37] is applied.

64048 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0318 [972] is applied.

64049 **Issue 8**

64050 Austin Group Defect 368 is applied, adding a statement that an unnamed semaphore may be
64051 implemented using a file descriptor and adding the [EMFILE] and [ENFILE] errors.

64052 Austin Group Defect 1216 is applied, adding *sem_clockwait*( ).

64053 **NAME**

64054 sem_open — initialize and open a named semaphore

64055 **SYNOPSIS**

64056 `#include <semaphore.h>`

64057 `sem_t *sem_open(const char *name, int oflag, ...);`

64058 **DESCRIPTION**

64059 The *sem_open*( ) function shall establish a connection between a named semaphore and a process.
64060 A named semaphore may be implemented using a file descriptor. Following a call to *sem_open*( )
64061 with semaphore name *name*, the process may reference the semaphore associated with *name*
64062 using the semaphore handle returned from the call. This semaphore can be used in subsequent
64063 calls to *sem_clockwait*( ), *sem_close*( ), *sem_post*( ), *sem_timedwait*( ), *sem_trywait*( ), and *sem_wait*( ).
64064 The semaphore remains usable by this process until the semaphore is closed by a successful call
64065 to *sem_close*( ), *_exit*( ), or one of the *exec* functions.

64066 The *oflag* argument controls whether the semaphore is created or merely accessed by the call to
64067 *sem_open*( ). The following flag bits may be set in *oflag*:

64068 O_CREAT This flag is used to create a semaphore if it does not already exist. If O_CREAT is
64069 set and the semaphore already exists, then O_CREAT has no effect, except as noted
64070 under O_EXCL. Otherwise, *sem_open*( ) creates a named semaphore. The O_CREAT
64071 flag requires a third and a fourth argument: *mode*, which is of type **mode_t**, and
64072 *value*, which is of type **unsigned**. The semaphore is created with an initial value of
64073 *value*. Valid initial values for semaphores are less than or equal to
64074 {SEM_VALUE_MAX}.

64075 The user ID of the semaphore shall be set to the effective user ID of the process.
64076 The group ID of the semaphore shall be set to the effective group ID of the process;
64077 however, if the *name* argument is visible in the file system, the group ID may be set
64078 to the group ID of the containing directory. The permission bits of the semaphore
64079 are set to the value of the *mode* argument except those set in the file mode creation
64080 mask of the process. When bits in *mode* other than the file permission bits are
64081 specified, the effect is unspecified.

64082 After the semaphore named *name* has been created by *sem_open*( ) with the
64083 O_CREAT flag, other processes can connect to the semaphore by calling
64084 *sem_open*( ) with the same value of *name*.

64085 O_EXCL If O_EXCL and O_CREAT are set, *sem_open*( ) fails if the semaphore *name* exists.
64086 The check for the existence of the semaphore and the creation of the semaphore if it
64087 does not exist are atomic with respect to other processes executing *sem_open*( ) with
64088 O_EXCL and O_CREAT set. If O_EXCL is set and O_CREAT is not set, the effect is
64089 undefined.

64090 If flags other than O_CREAT and O_EXCL are specified in the *oflag* parameter, the
64091 effect is unspecified.

64092 The *name* argument points to a string naming a semaphore object. It is unspecified whether the
64093 name appears in the file system and is visible to functions that take pathnames as arguments.
64094 The *name* argument conforms to the construction rules for a pathname, except that the
64095 interpretation of <slash> characters other than the leading <slash> character in *name* is
64096 implementation-defined, and that the length limits for the *name* argument are implementation-
64097 defined and need not be the same as the pathname limits {PATH_MAX} and {NAME_MAX}. If
64098 *name* begins with the <slash> character, then processes calling *sem_open*( ) with the same value of
64099 *name* shall refer to the same semaphore object, as long as that name has not been removed. If

64100 *name* does not begin with the <slash> character, the effect is implementation-defined.

64101 If a process makes multiple successful calls to *sem_open*( ) with the same value for *name*, there
64102 have been no intervening calls to *sem_unlink*( ) for *name*, and at least one open handle for this
64103 semaphore has not been closed with a *sem_close*( ) call, it is implementation-defined whether the
64104 same handle or a unique handle is returned for each such successful call.

64105 References to copies of the semaphore produce undefined results.

64106 **RETURN VALUE**

64107 Upon successful completion, the *sem_open*( ) function shall return the address of the semaphore.
64108 Otherwise, it shall return a value of SEM_FAILED and set *errno* to indicate the error. The symbol
64109 SEM_FAILED is defined in the **<semaphore.h>** header. No successful return from *sem_open*( )
64110 shall return the value SEM_FAILED.

64111 **ERRORS**

64112 If any of the following conditions occur, the *sem_open*( ) function shall return SEM_FAILED and
64113 set *errno* to the corresponding value:

64114 [EACCES] The named semaphore exists and the permissions specified by *oflag* are
64115 denied, or the named semaphore does not exist and permission to create the
64116 named semaphore is denied.

64117 [EEXIST] O_CREAT and O_EXCL are set and the named semaphore already exists.

64118 [EINTR] The *sem_open*( ) operation was interrupted by a signal.

64119 [EINVAL] The *sem_open*( ) operation is not supported for the given name, or O_CREAT
64120 was specified in *oflag* and *value* was greater than {SEM_VALUE_MAX}.

64121 [ENOENT] O_CREAT is not set and the named semaphore does not exist.

64122 [ENOMEM] There is insufficient memory for the creation of the new named semaphore.

64123 [ENOSPC] There is insufficient space on a storage device for the creation of the new
64124 named semaphore.

64125 If any of the following conditions occur, the *sem_open*( ) function may return SEM_FAILED and
64126 set *errno* to the corresponding value:

64127 [EMFILE] Too many semaphore descriptors or file descriptors are currently in use by this
64128 process.

64129 [ENAMETOOLONG]

64130 The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems
64131 XSI that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI
64132 systems, or has a pathname component that is longer than
64133 XSI {_POSIX_NAME_MAX} on systems that do not support the XSI option or
64134 longer than {_XOPEN_NAME_MAX} on XSI systems.

64135 [ENFILE] Too many semaphore descriptors or file descriptors are currently open in the
64136 system.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Early drafts required an error return value of −1 with the type **sem_t *** for the *sem_open*( ) function, which is not guaranteed to be portable across implementations. The revised text provides the symbolic error code SEM_FAILED to eliminate the type conflict.

FUTURE DIRECTIONS

A future version might require the *sem_open*( ) and *sem_unlink*( ) functions to have semantics similar to normal file system operations.

SEE ALSO

semctl( ), *semget*( ), *semop*( ), *sem_clockwait*( ), *sem_close*( ), *sem_post*( ), *sem_trywait*( ), *sem_unlink*( )

XBD **<semaphore.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *sem_open*( ) function is marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

The *sem_timedwait*( ) function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/117 is applied, updating the DESCRIPTION to add the *sem_timedwait*( ) function for alignment with IEEE Std 1003.1d-1999.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/118 is applied, updating the DESCRIPTION to describe the conditions to return the same semaphore address on a call to *sem_open*( ). The words ``and at least one previous successful *sem_open*( ) call for this semaphore has not been matched with a *sem_close*( ) call'' are added.

Issue 7

Austin Group Interpretation 1003.1-2001 #066 is applied, updating the [ENOSPC] error case and adding the [ENOMEM] error case.

Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and adding [ENAMETOOLONG] as a ``may fail'' error.

Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the user ID and group ID of the semaphore.

The *sem_open*( ) function is moved from the Semaphores option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0527 [37] is applied.

Issue 8

Austin Group Defect 368 is applied, adding a statement that a named semaphore may be implemented using a file descriptor and changing the ERRORS section.

Austin Group Defect 1216 is applied, adding *sem_clockwait*( ).

Austin Group Defect 1324 is applied, making it implementation-defined whether the same handle or a unique handle is returned when multiple successful calls to *sem_open*( ) are made with the same value for *name*.

64182 **NAME**

64183 sem_post — unlock a semaphore

64184 **SYNOPSIS**

64185 `#include <semaphore.h>`

64186 `int sem_post(sem_t *sem);`

64187 **DESCRIPTION**

64188 The *sem_post*( ) function shall unlock the semaphore referenced by *sem* by performing a
64189 semaphore unlock operation on that semaphore.

64190 If the semaphore value resulting from this operation is positive, then no threads were blocked
64191 waiting for the semaphore to become unlocked; the semaphore value is simply incremented.

64192 If the value of the semaphore resulting from this operation is zero, then one of the threads
64193 blocked waiting for the semaphore shall be allowed to return successfully from its call to
64194 PS *sem_wait*( ). If the Process Scheduling option is supported, the thread to be unblocked shall be
64195 chosen in a manner appropriate to the scheduling policies and parameters in effect for the
64196 blocked threads. In the case of the schedulers SCHED_FIFO and SCHED_RR, the highest
64197 priority waiting thread shall be unblocked, and if there is more than one highest priority thread
64198 blocked waiting for the semaphore, then the highest priority thread that has been waiting the
64199 longest shall be unblocked. If the Process Scheduling option is not defined, the choice of a thread
64200 to unblock is unspecified.

64201 SS If the Process Sporadic Server option is supported, and the scheduling policy is
64202 SCHED_SPORADIC, the semantics are as per SCHED_FIFO above.

64203 The *sem_post*( ) function shall be async-signal-safe and may be invoked from a signal-catching
64204 function.

64205 **RETURN VALUE**

64206 If successful, the *sem_post*( ) function shall return zero; otherwise, the function shall return –1
64207 and set *errno* to indicate the error.

64208 **ERRORS**

64209 The *sem_post*( ) function shall fail if:

64210 [EOVERFLOW] The maximum allowable value of the semaphore would be exceeded.

64211 The *sem_post*( ) function may fail if:

64212 [EINVAL] The *sem* argument does not refer to a valid semaphore.

64213 **EXAMPLES**

64214 See *sem_clockwait*( ).

64215 **APPLICATION USAGE**

64216 None.

64217 **RATIONALE**

64218 None.

64219 **FUTURE DIRECTIONS**

64220 None.

64221 **SEE ALSO**

64222 *semctl*( ), *semget*( ), *semop*( ), *sem_clockwait*( ), *sem_trywait*( )

64223 XBD Section 4.15.2 (on page 104), **<semaphore.h>**

64224 **CHANGE HISTORY**

64225 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

64226 **Issue 6**

64227 The *sem_post*( ) function is marked as part of the Semaphores option.

64228 The [ENOSYS] error condition has been removed as stubs need not be provided if an
64229 implementation does not support the Semaphores option.

64230 The *sem_timedwait*( ) function is added to the SEE ALSO section for alignment with IEEE Std
64231 1003.1d-1999.

64232 SCHED_SPORADIC is added to the list of scheduling policies for which the thread that is to be
64233 unblocked is specified for alignment with IEEE Std 1003.1d-1999.

64234 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/119 is applied, updating the ERRORS
64235 section so that the [EINVAL] error becomes optional.

64236 **Issue 7**

64237 Austin Group Interpretation 1003.1-2001 #156 is applied.

64238 The *sem_post*( ) function is moved from the Semaphores option to the Base.

64239 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0528 [37] is applied.

64240 **Issue 8**

64241 Austin Group Defect 315 is applied, adding the [EOVERFLOW] error.

64242 **NAME**

64243 sem_timedwait — lock a semaphore

64244 **SYNOPSIS**

64245 ```
#include <semaphore.h>
```

64246 ```
int sem_timedwait(sem_t *restrict sem,
```

64247 ```
    const struct timespec *restrict abstime);
```

64248 **DESCRIPTION**

64249 Refer to *sem_clockwait( )*.

  
```

NAME

sem_trywait, sem_wait — lock a semaphore

SYNOPSIS

```
#include <semaphore.h>

int sem_trywait(sem_t *sem);
int sem_wait(sem_t *sem);
```

DESCRIPTION

The *sem_trywait*( ) function shall lock the semaphore referenced by *sem* only if the semaphore is currently not locked; that is, if the semaphore value is currently positive. Otherwise, it shall not lock the semaphore.

The *sem_wait*( ) function shall lock the semaphore referenced by *sem* by performing a semaphore lock operation on that semaphore. If the semaphore value is currently zero, then the calling thread shall not return from the call to *sem_wait*( ) until it either locks the semaphore or the call is interrupted by a signal.

Upon successful return, the state of the semaphore shall be locked and shall remain locked until the *sem_post*( ) function is executed and returns successfully.

The *sem_wait*( ) function is interruptible by the delivery of a signal.

RETURN VALUE

The *sem_trywait*( ) and *sem_wait*( ) functions shall return zero if the calling process successfully performed the semaphore lock operation on the semaphore designated by *sem*. If the call was unsuccessful, the state of the semaphore shall be unchanged, and the function shall return a value of –1 and set *errno* to indicate the error.

ERRORS

The *sem_trywait*( ) function shall fail if:

[EAGAIN] The semaphore was already locked, so it cannot be immediately locked by the *sem_trywait*( ) operation.

The *sem_trywait*( ) and *sem_wait*( ) functions may fail if:

[EDEADLK] A deadlock condition was detected.

[EINTR] A signal interrupted this function.

[EINVAL] The *sem* argument does not refer to a valid semaphore.

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in XBD Section 3.275 (on page 72).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

semctl( ), *semget*( ), *semop*( ), *sem_clockwait*( ), *sem_post*( )

XBD Section 3.275 (on page 72), Section 4.15.2 (on page 104), **<semaphore.h>**

64292 **CHANGE HISTORY**

64293 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

64294 **Issue 6**

64295 The *sem_trywait*( ) and *sem_wait*( ) functions are marked as part of the Semaphores option.

64296 The [ENOSYS] error condition has been removed as stubs need not be provided if an
64297 implementation does not support the Semaphores option.

64298 The *sem_timedwait*( ) function is added to the SEE ALSO section for alignment with IEEE Std
64299 1003.1d-1999.

64300 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/121 is applied, updating the ERRORS
64301 section so that the [EINVAL] error becomes optional.

64302 **Issue 7**

64303 SD5-XSH-ERN-54 is applied, removing the *sem_wait*( ) function from the ``shall fail'' error cases.

64304 The *sem_trywait*( ) and *sem_wait*( ) functions are moved from the Semaphores option to the Base.

64305 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0530 [37] is applied.

NAME

sem_unlink — remove a named semaphore

SYNOPSIS

```
#include <semaphore.h>

int sem_unlink(const char *name);
```

DESCRIPTION

The *sem_unlink*( ) function shall remove the semaphore named by the string *name*. If the semaphore named by *name* is currently referenced by other processes, then *sem_unlink*( ) shall have no effect on the state of the semaphore. If one or more processes have the semaphore open when *sem_unlink*( ) is called, destruction of the semaphore is postponed until all references to the semaphore have been destroyed by calls to *sem_close*( ), *_exit*( ), or *exec*. Calls to *sem_open*( ) to recreate or reconnect to the semaphore refer to a new semaphore after *sem_unlink*( ) is called. The *sem_unlink*( ) call shall not block until all references have been destroyed; it shall return immediately.

RETURN VALUE

Upon successful completion, the *sem_unlink*( ) function shall return a value of 0. Otherwise, the semaphore shall not be changed and the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *sem_unlink*( ) function shall fail if:

[EACCES] Permission is denied to unlink the named semaphore.

[ENOENT] The named semaphore does not exist.

The *sem_unlink*( ) function may fail if:

[ENAMETOOLONG]
The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems that do not support the XSI option (XSI) or exceeds {_XOPEN_PATH_MAX} on XSI systems, or has a pathname component that is longer than {_POSIX_NAME_MAX} on systems that do not support the XSI option (XSI) or longer than {_XOPEN_NAME_MAX} on XSI systems. A call to *sem_unlink*( ) with a *name* argument that contains the same semaphore name as was previously used in a successful *sem_open*( ) call shall not give an [ENAMETOOLONG] error.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

A future version might require the *sem_open*( ) and *sem_unlink*( ) functions to have semantics similar to normal file system operations.

SEE ALSO

semctl( ), *semget*( ), *semop*( ), *sem_close*( ), *sem_open*( )

XBD **<semaphore.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *sem_unlink*( ) function is marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

Issue 7

Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a ``shall fail'' to a ``may fail'' error.

Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

The *sem_unlink*( ) function is moved from the Semaphores option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0531 [37] is applied.

64362 **NAME**

64363 sem_wait — lock a semaphore

64364 **SYNOPSIS**

64365 `#include <semaphore.h>`

64366 `int sem_wait(sem_t *sem);`

64367 **DESCRIPTION**

64368 Refer to *sem_trywait( )*.

NAME

semctl — XSI semaphore control operations

SYNOPSIS

XSI
```
#include <sys/sem.h>

int semctl(int semid, int semnum, int cmd, ...);
```

DESCRIPTION

The *semctl*( ) function operates on XSI semaphores (see XBD Section 4.20, on page 108). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 527).

The *semctl*( ) function provides a variety of semaphore control operations as specified by *cmd*. The fourth argument is optional and depends upon the operation requested. If required, it is of type **union semun**, which the application shall explicitly declare:

```
union semun {
    int val;
    struct semid_ds *buf;
    unsigned short  *array;
} arg;
```

Each operation shall be performed atomically.

The following semaphore control operations as specified by *cmd* are executed with respect to the semaphore specified by *semid* and *semnum*. The level of permission required for each operation is shown with each command; see Section 2.7 (on page 526). The symbolic names for the values of *cmd* are defined in the **<sys/sem.h>** header:

GETVAL — Return the value of *semval*; see **<sys/sem.h>**. Requires read permission.

SETVAL — Set the value of *semval* to *arg.val*, where *arg* is the value of the fourth argument to *semctl*( ). When this command is successfully executed, the *semadj* value corresponding to the specified semaphore in all processes is cleared. Also, the *sem_ctime* timestamp shall be set to the current time, as described in Section 2.7.1 (on page 526). Requires alter permission; see Section 2.7 (on page 526).

GETPID — Return the value of *sempid*. Requires read permission.

GETNCNT — Return the value of *semncnt*. Requires read permission.

GETZCNT — Return the value of *semzcnt*. Requires read permission.

The following values of *cmd* operate on each *semval* in the set of semaphores:

GETALL — Return the value of *semval* for each semaphore in the semaphore set and place into the array pointed to by *arg.array*, where *arg* is the fourth argument to *semctl*( ). Requires read permission.

SETALL — Set the value of *semval* for each semaphore in the semaphore set according to the array pointed to by *arg.array*, where *arg* is the fourth argument to *semctl*( ). When this command is successfully executed, the *semadj* values corresponding to each specified semaphore in all processes are cleared. Also, the *sem_ctime* timestamp shall be set to the current time, as described in Section 2.7.1 (on page 526). Requires alter permission.

The following values of *cmd* are also available:

IPC_STAT — Place the current value of each member of the **semid_ds** data structure associated with *semid* into the structure pointed to by *arg.buf*, where *arg* is the fourth argument to *semctl*( ). The contents of this structure are defined in **<sys/sem.h>**. Requires read permission.

IPC_SET — Set the value of the following members of the **semid_ds** data structure associated with *semid* to the corresponding value found in the structure pointed to by *arg.buf*, where *arg* is the fourth argument to *semctl*( ):

```
sem_perm.uid
sem_perm.gid
sem_perm.mode
```

The mode bits specified in Section 2.7.1 (on page 526) are copied into the corresponding bits of the *sem_perm.mode* associated with *semid*. The stored values of any other bits are unspecified. The *sem_ctime* timestamp shall be set to the current time, as described in Section 2.7.1 (on page 526).

This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of *sem_perm.cuid* or *sem_perm.uid* in the **semid_ds** data structure associated with *semid*.

IPC_RMID — Remove the semaphore identifier specified by *semid* from the system and destroy the set of semaphores and **semid_ds** data structure associated with it. This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of *sem_perm.cuid* or *sem_perm.uid* in the **semid_ds** data structure associated with *semid*.

RETURN VALUE

If successful, the value returned by *semctl*( ) depends on *cmd* as follows:

GETVAL — The value of *semval*.

GETPID — The value of *sempid*.

GETNCNT — The value of *semncnt*.

GETZCNT — The value of *semzcnt*.

All others — 0.

Otherwise, *semctl*( ) shall return –1 and set *errno* to indicate the error.

ERRORS

The *semctl*( ) function shall fail if:

[EACCES] — Operation permission is denied to the calling process; see Section 2.7 (on page 526).

[EINVAL] — The value of *semid* is not a valid semaphore identifier, or the value of *semnum* is less than 0 or greater than or equal to *sem_nsems*, or the value of *cmd* is not a valid command.

[EPERM] — The argument *cmd* is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of *sem_perm.cuid* or *sem_perm.uid* in the data structure associated with *semid*.

64454 [ERANGE] The argument *cmd* is equal to SETVAL or SETALL and the value to which
64455 *semval* is to be set is greater than the system-imposed maximum.

64456 **EXAMPLES**

64457 Refer to *semop( )*.

64458 **APPLICATION USAGE**

64459 The fourth parameter in the SYNOPSIS section is now specified as `"..."` in order to avoid a
64460 clash with the ISO C standard when referring to the union *semun* (as defined in Issue 3) and for
64461 backwards-compatibility.

64462 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
64463 Application developers who need to use IPC should design their applications so that modules
64464 using the IPC routines described in Section 2.7 (on page 526) can be easily modified to use the
64465 alternative interfaces.

64466 **RATIONALE**

64467 None.

64468 **FUTURE DIRECTIONS**

64469 None.

64470 **SEE ALSO**

64471 Section 2.7 (on page 526), Section 2.8 (on page 527), *semget( )*, *semop( )*, *sem_close( )*, *sem_destroy( )*,
64472 *sem_getvalue( )*, *sem_init( )*, *sem_open( )*, *sem_post( )*, *sem_trywait( )*, *sem_unlink( )*

64473 XBD Section 4.20 (on page 108), **<sys/sem.h>**

64474 **CHANGE HISTORY**

64475 First released in Issue 2. Derived from Issue 2 of the SVID.

64476 **Issue 5**

64477 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
64478 DIRECTIONS to the APPLICATION USAGE section.

64479 **Issue 7**

64480 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0532 [345], XSH/TC1-2008/0533 [345],
64481 XSH/TC1-2008/0534 [345], and XSH/TC1-2008/0535 [335] are applied.

64482 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0319 [532] is applied.

NAME

semget — get set of XSI semaphores

SYNOPSIS

XSI
```
#include <sys/sem.h>

int semget(key_t key, int nsems, int semflg);
```

DESCRIPTION

The *semget*( ) function operates on XSI semaphores (see XBD Section 4.20, on page 108). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 527).

The *semget*( ) function shall return the semaphore identifier associated with *key*.

A semaphore identifier with its associated **semid_ds** data structure and its associated set of *nsems* semaphores (see **<sys/sem.h>**) is created for *key* if one of the following is true:

- The argument *key* is equal to IPC_PRIVATE.
- The argument *key* does not already have a semaphore identifier associated with it and (*semflg* &IPC_CREAT) is non-zero.

Upon creation, the **semid_ds** data structure associated with the new semaphore identifier is initialized as follows:

- In the operation permissions structure *sem_perm.cuid*, *sem_perm.uid*, *sem_perm.cgid*, and *sem_perm.gid* shall be set to the effective user ID and effective group ID, respectively, of the calling process.
- The low-order 9 bits of *sem_perm.mode* shall be set to the low-order 9 bits of *semflg*.
- The variable *sem_nsems* shall be set to the value of *nsems*.
- The variable *sem_otime* shall be set to 0 and *sem_ctime* shall be set to the current time, as described in Section 2.7.1 (on page 526).

Upon creation, the value of the *semval*, *sempid*, *semncnt*, and *semzcnt* members of all *nsems* semaphores in the associated semaphore set shall be set to zero.

RETURN VALUE

Upon successful completion, *semget*( ) shall return a non-negative integer, namely a semaphore identifier; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *semget*( ) function shall fail if:

[EACCES] A semaphore identifier exists for *key*, but operation permission as specified by the low-order 9 bits of *semflg* would not be granted; see Section 2.7 (on page 526).

[EEXIST] A semaphore identifier exists for the argument *key* but ((*semflg* &IPC_CREAT) &&(*semflg* &IPC_EXCL)) is non-zero.

[EINVAL] The value of *nsems* is either less than or equal to 0 or greater than the system-imposed limit, or a semaphore identifier exists for the argument *key*, but the number of semaphores in the set associated with it is less than *nsems* and *nsems* is not equal to 0.

[ENOENT] A semaphore identifier does not exist for the argument *key* and (*semflg* &IPC_CREAT) is equal to 0.

[ENOSPC] A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphores system-wide would be exceeded.

EXAMPLES

Refer to *semop*( ).

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 526) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7 (on page 526), Section 2.8 (on page 527), *ftok*( ), *semctl*( ), *semop*( ), *sem_close*( ), *sem_destroy*( ), *sem_getvalue*( ), *sem_init*( ), *sem_open*( ), *sem_post*( ), *sem_trywait*( ), *sem_unlink*( )

XBD Section 4.20 (on page 108), **<sys/sem.h>**

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

Issue 6

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/122 is applied, updating the DESCRIPTION from ``each semaphore in the set shall not be initialized'' to ``each semaphore in the set need not be initialized''.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0536 [335,439] and XSH/TC1-2008/0537 [344] are applied.

Issue 8

Austin Group Defect 377 is applied, adding a requirement that the value of the *semval*, *sempid*, *semncnt*, and *semzcnt* members of all semaphores in a semaphore set be initialized to zero when a call to *semget*( ) creates a semaphore set.

NAME

semop — XSI semaphore operations

SYNOPSIS

XSI
```
#include <sys/sem.h>

int semop(int semid, struct sembuf *sops, size_t nsops);
```

DESCRIPTION

The *semop*( ) function operates on XSI semaphores (see XBD Section 4.20, on page 108). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 527).

The *semop*( ) function shall perform atomically a user-defined array of semaphore operations in array order on the set of semaphores associated with the semaphore identifier specified by the argument *semid*.

The argument *sops* is a pointer to a user-defined array of semaphore operation structures. The implementation shall not modify elements of this array unless the application uses implementation-defined extensions.

The argument *nsops* is the number of such structures in the array.

Each structure, **sembuf**, includes the following members:

Member Type	Member Name	Description
unsigned short	*sem_num*	Semaphore number.
short	*sem_op*	Semaphore operation.
short	*sem_flg*	Operation flags.

Each semaphore operation specified by *sem_op* is performed on the corresponding semaphore specified by *semid* and *sem_num*.

If all of the semaphore operations complete successfully, *semop*( ) shall return 0. If a semaphore operation fails or blocks, all changes to *semadj* and *semval* values performed by completed semaphore operations in the array before the operation that failed or blocked shall be undone before *semop*( ) returns or blocks, respectively. When a semaphore operation blocks, the change to *semncnt* or *semzcnt* indicating which semaphore operation blocked shall not be undone until the operation unblocks.

For each operation in the array of semaphore operations, the variable *sem_op* specifies one of three semaphore operations:

1. If *sem_op* is a negative integer and the calling process has alter permission, one of the following shall occur:
   - If *semval* (see **<sys/sem.h>**) is greater than or equal to the absolute value of *sem_op*, the absolute value of *sem_op* shall be subtracted from *semval*. Also, if (*sem_flg* & SEM_UNDO) is non-zero, the absolute value of *sem_op* shall be added to the *semadj* value of the calling process for the specified semaphore. If this is not the last operation in the array of semaphore operations to be performed, processing shall continue with the next operation in the array.
   - If *semval* is less than the absolute value of *sem_op* and (*sem_flg* & IPC_NOWAIT) is non-zero, the operation shall fail with *errno* set to [EAGAIN].

- If *semval* is less than the absolute value of *sem_op* and (*sem_flg* & IPC_NOWAIT) is 0, *semop*( ) shall increment the *semncnt* associated with the specified semaphore and suspend execution of the calling thread (block) until one of the following conditions occurs:
  - — The value of *semval* changes. When this occurs, the value of *semncnt* associated with the specified semaphore shall be decremented and the array of semaphore operations shall be reevaluated.
  - — The *semid* for which the calling thread is awaiting action is removed from the system. When this occurs, the operation shall fail with *errno* set to [EIDRM].
  - — The calling thread receives a signal that is to be caught. When this occurs, the value of *semncnt* associated with the specified semaphore shall be decremented, and the calling thread shall resume execution in the manner prescribed in *sigaction*( ).

2. If *sem_op* is a positive integer and the calling process has alter permission, the value of *sem_op* shall be added to *semval* and, if (*sem_flg* & SEM_UNDO) is non-zero, the value of *sem_op* shall be subtracted from the *semadj* value of the calling process for the specified semaphore. If this is not the last operation in the array of semaphore operations to be performed, processing shall continue with the next operation in the array.
3. If *sem_op* is 0 and the calling process has read permission, one of the following shall occur:
   - If *semval* is 0, this is a successful operation. If this is not the last operation in the array of semaphore operations to be performed, processing shall continue with the next operation in the array.
   - If *semval* is non-zero and (*sem_flg* & IPC_NOWAIT) is non-zero, the operation shall fail with *errno* set to [EAGAIN].
   - If *semval* is non-zero and (*sem_flg* & IPC_NOWAIT) is 0, *semop*( ) shall increment the *semzcnt* associated with the specified semaphore and suspend execution of the calling thread (block) until one of the following occurs:
     - — The value of *semval* changes. When this occurs, the value of *semzcnt* associated with the specified semaphore shall be decremented and the array of semaphore operations shall be reevaluated.
     - — The *semid* for which the calling thread is awaiting action is removed from the system. When this occurs, the operation shall fail with *errno* set to [EIDRM].
     - — The calling thread receives a signal that is to be caught. When this occurs, the value of *semzcnt* associated with the specified semaphore shall be decremented, and the calling thread shall resume execution in the manner prescribed in *sigaction*( ).

Upon successful completion of all of the semaphore operations specified in the array pointed to by *sops*, the value of *sempid* for each semaphore specified in the array shall be set to the process ID of the calling process and the *sem_otime* timestamp associated with the semaphore set shall be set to the current time, as described in Section 2.7.1 (on page 526).

RETURN VALUE

Upon successful completion, *semop*( ) shall return 0; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *semop*( ) function shall fail if:

[E2BIG] The value of *nsops* is greater than the system-imposed maximum.

[EACCES] Operation permission is denied to the calling process; see Section 2.7 (on page 526).

[EAGAIN] The operation would result in suspension of the calling thread but (*sem_flg* & IPC_NOWAIT) is non-zero.

[EFBIG] The value of *sem_num* is greater than or equal to the number of semaphores in the set associated with *semid*.

[EIDRM] The semaphore identifier *semid* is removed from the system.

[EINTR] The *semop*( ) function was interrupted by a signal.

[EINVAL] The value of *semid* is not a valid semaphore identifier, or the number of individual semaphores for which the calling process requests a SEM_UNDO would exceed the system-imposed limit.

[ENOSPC] The limit on the number of individual processes requesting a SEM_UNDO would be exceeded.

[ERANGE] An operation would cause a *semval* to overflow the system-imposed limit, or an operation would cause a *semadj* value to overflow the system-imposed limit.

EXAMPLES

Setting Values in Semaphores

The following example sets the values of the two semaphores associated with the *semid* identifier to the values contained in the *sb* array.

```
#include <sys/sem.h>
...
int semid;
struct sembuf sb[2];
int nsops = 2;
int result;

// Code to initialize semid.
...

// Adjust value of semaphore in the semaphore array semid.
sb[0].sem_num = 0;
sb[0].sem_op = -1;
sb[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
sb[1].sem_num = 1;
sb[1].sem_op = 1;
sb[1].sem_flg = 0;

result = semop(semid, sb, nsops);
```


Creating a Semaphore Identifier

The following example gets a semaphore key using the *ftok*( ) function, then creates or uses an existing semaphore set associated with that key using the *semget*( ) function.

If this process creates the semaphore set, the program uses a call to *semop*( ) to initialize it to the value in the *sbuf* array. The number of processes that can execute concurrently is set to 2.

The final call to *semop*( ) acquires the semaphore and waits until it is free; the SEM_UNDO option releases the semaphore when the process exits, waiting until there are less than two processes running concurrently.

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/sem.h>
#include <sys/stat.h>
...
struct sembuf sbuf;
int semid;
key_t semkey;
...
// Get a key for the semaphore set.
if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
    perror("IPC error: ftok");
    exit(1);
}

// Create the semaphore set associated with this key
if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
    S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1) {

    // Initialize the semaphore.
    sbuf.sem_num = 0;
    sbuf.sem_op = 2; // Set the number of runs without queuing.
    sbuf.sem_flg = 0;
    if (semop(semid, &sbuf, 1) == -1) {
       perror("IPC error: semop");
       exit(1);
    }
} else if (errno == EEXIST) {

    // The semaphore set already exists; get its semaphore ID.
    if ((semid = semget(semkey, 0, 0)) == -1) {
       perror("IPC error 1: semget");
       exit(1);
    }
} else {
    perror("IPC error 2: semget");
    exit(1);
}

// Since the semget() initialized the semaphore to 0, the
// following semop() will block until the creating process
```



```
        // completes the initialization above. Processes will also
        // block in the following semop() call if two other processes
        // have already passed this point and are still running.
        sbuf.sem_num = 0;
        sbuf.sem_op = -1;
        sbuf.sem_flg = SEM_UNDO;
        if (semop(semid, &sbuf, 1) == -1) {
            perror("IPC Error: semop");
            exit(1);
        }
```

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 526) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7 (on page 526), Section 2.8 (on page 527), *exec*, *exit*( ), *fork*( ), *semctl*( ), *semget*( ), *sem_close*( ), *sem_destroy*( ), *sem_getvalue*( ), *sem_init*( ), *sem_open*( ), *sem_post*( ), *sem_trywait*( ), *sem_unlink*( )

XBD Section 4.20 (on page 108), **<sys/ipc.h>**, **<sys/sem.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

Issue 7

SD5-XSH-ERN-171 is applied, updating the DESCRIPTION to clarify the order in which the operations in *sops* will be performed when there are multiple operations.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0538 [329,429], XSH/TC1-2008/0539 [345,428], XSH/TC1-2008/0540 [329,429], XSH/TC1-2008/0541 [335], and XSH/TC1-2008/0542 [291,429] are applied.

Issue 8

Austin Group Defect 377 is applied, changing the EXAMPLES section.

Austin Group Defect 628 is applied, clarifying how *semop*( ) behaves when there is more than one operation in the array specified by *sops*.

64766 **NAME**

64767 send — send a message on a socket

64768 **SYNOPSIS**

64769 `#include <sys/socket.h>`

64770 `ssize_t send(int socket, const void *buffer, size_t length, int flags);`

64771 **DESCRIPTION**

64772 The *send*( ) function shall initiate transmission of a message from the specified socket to its peer.
64773 The *send*( ) function shall send a message only when the socket is connected. If the socket is a
64774 connectionless-mode socket, the message shall be sent to the pre-specified peer address.

64775 The *send*( ) function takes the following arguments:

64776 *socket* Specifies the socket file descriptor.

64777 *buffer* Points to the buffer containing the message to send.

64778 *length* Specifies the length of the message in bytes.

64779 *flags* Specifies the type of message transmission. Values of this argument are
64780 formed by logically OR'ing zero or more of the following flags:

64781 MSG_EOR Terminates a record (if supported by the protocol).

64782 MSG_OOB Sends out-of-band data on sockets that support out-of-
64783 band communications. The significance and semantics
64784 of out-of-band data are protocol-specific.

64785 MSG_NOSIGNAL Requests not to send the SIGPIPE signal if an attempt to
64786 send is made on a stream-oriented socket that is no
64787 longer connected. The [EPIPE] error shall still be
64788 returned.

64789 The length of the message to be sent is specified by the *length* argument. If the message is too
64790 long to pass through the underlying protocol, *send*( ) shall fail and no data shall be transmitted.

64791 Successful completion of a call to *send*( ) does not guarantee delivery of the message. A return
64792 value of –1 indicates only locally-detected errors.

64793 If space is not available at the sending socket to hold the message to be transmitted, and the
64794 socket file descriptor does not have O_NONBLOCK set, *send*( ) shall block until space is
64795 available or a timeout occurs (see SO_SNDTIMEO in Section 2.10.16, on page 554). If space is
64796 not available at the sending socket to hold the message to be transmitted, and the socket file
64797 descriptor does have O_NONBLOCK set, *send*( ) shall fail. The *select*( ) and *poll*( ) functions can
64798 be used to determine when it is possible to send more data.

64799 The socket in use may require the process to have appropriate privileges to use the *send*( )
64800 function.

64801 **RETURN VALUE**

64802 Upon successful completion, *send*( ) shall return the number of bytes sent. Otherwise, –1 shall be
64803 returned and *errno* set to indicate the error.

64804 **ERRORS**

64805 The *send*( ) function shall fail if:

64806 [EAGAIN] or [EWOULDBLOCK]
64807 The socket's file descriptor is marked O_NONBLOCK and the requested
64808 operation would block. See also SO_SNDTIMEO in Section 2.10.16 (on page
64809 554).

64810 [EBADF] The *socket* argument is not a valid file descriptor.

64811 [ECONNRESET] A connection was forcibly closed by a peer.

64812 [EDESTADDRREQ]
64813 The socket is not connection-mode and no peer address is set.

64814 [EINTR] A signal interrupted *send*( ) before any data was transmitted.

64815 [EMSGSIZE] The message is too large to be sent all at once, as the socket requires.

64816 [ENOTCONN] The socket is not connected.

64817 [ENOTSOCK] The *socket* argument does not refer to a socket.

64818 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or
64819 more of the values set in *flags*.

64820 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is
64821 no longer connected. In the latter case, and if the socket is of type
64822 SOCK_STREAM or SOCK_SEQPACKET and the MSG_NOSIGNAL flag is not
64823 set, the SIGPIPE signal is generated to the calling thread.

64824 The *send*( ) function may fail if:

64825 [EACCES] The calling process does not have appropriate privileges.

64826 [EIO] An I/O error occurred while reading from or writing to the file system.

64827 [ENETDOWN] The local network interface used to reach the destination is down.

64828 [ENETUNREACH]
64829 No route to the network is present.

64830 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

64831 **EXAMPLES**

64832 None.

64833 **APPLICATION USAGE**

64834 If the *socket* argument refers to a connection-mode socket, the *send*( ) function is equivalent to
64835 *sendto*( ) (with any value for the *dest_addr* and *dest_len* arguments, as they are ignored in this
64836 case). If the *socket* argument refers to a socket and the *flags* argument is 0, the *send*( ) function is
64837 equivalent to *write*( ).

64838 **RATIONALE**

64839 None.

64840 **FUTURE DIRECTIONS**

64841 None.

64842 **SEE ALSO**

64843 *connect*( ), *getsockopt*( ), *poll*( ), *pselect*( ), *recv*( ), *recvfrom*( ), *recvmsg*( ), *sendmsg*( ), *sendto*( ),
64844 *setsockopt*( ), *shutdown*( ), *socket*( ), *write*( )

64845 XBD **<sys/socket.h>**

64846 **CHANGE HISTORY**

64847 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

64848 **Issue 7**

64849 Austin Group Interpretation 1003.1-2001 #035 is applied, updating the DESCRIPTION to clarify
64850 the behavior when the socket is a connectionless-mode socket.

64851 The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended
64852 API Set Part 2.

64853 The [EPIPE] error is modified.

64854 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0543 [463] is applied.

64855 **Issue 8**

64856 Austin Group Defect 1429 is applied, clarifying the behavior on timeout by adding references to
64857 Section 2.10.16 (on page 554).

64858 **NAME**

64859 sendmsg — send a message on a socket using a message structure

64860 **SYNOPSIS**

64861 ```
#include <sys/socket.h>
```

64862 ```
ssize_t sendmsg(int socket, const struct msghdr *message, int flags);
```

64863 **DESCRIPTION**

64864 The *sendmsg*( ) function shall send a message through a connection-mode or connectionless-
64865 mode socket. If the socket is a connectionless-mode socket, the message shall be sent to the
64866 address specified by **msghdr** if no pre-specified peer address has been set. If a peer address has
64867 been pre-specified, either the message shall be sent to the address specified in **msghdr**
64868 (overriding the pre-specified peer address), or the function shall return –1 and set *errno* to
64869 [EISCONN]. If the socket is connection-mode, the destination address in **msghdr** shall be
64870 ignored.

64871 The *sendmsg*( ) function takes the following arguments:

64872 *socket* Specifies the socket file descriptor.

64873 *message* Points to a **msghdr** structure, containing both the destination address and the
64874 buffers for the outgoing message. The length and format of the address
64875 depend on the address family of the socket. The *msg_flags* member is ignored.

64876 *flags* Specifies the type of message transmission. The application may specify 0 or
64877 the following flag:

64878 MSG_EOR Terminates a record (if supported by the protocol).

64879 MSG_OOB Sends out-of-band data on sockets that support out-of-
64880 bound data. The significance and semantics of out-of-
64881 band data are protocol-specific.

64882 MSG_NOSIGNAL Requests not to send the SIGPIPE signal if an attempt to
64883 send is made on a stream-oriented socket that is no
64884 longer connected. The [EPIPE] error shall still be
64885 returned.

64886 The *msg_iov* and *msg_iovlen* fields of *message* specify zero or more buffers containing the data to
64887 be sent. *msg_iov* points to an array of **iovec** structures; *msg_iovlen* shall be set to the dimension of
64888 this array. In each **iovec** structure, the *iov_base* field specifies a storage area and the *iov_len* field
64889 gives its size in bytes. Some of these sizes can be zero. The data from each storage area indicated
64890 by *msg_iov* is sent in turn.

64891 Successful completion of a call to *sendmsg*( ) does not guarantee delivery of the message. A
64892 return value of –1 indicates only locally-detected errors.

64893 If space is not available at the sending socket to hold the message to be transmitted and the
64894 socket file descriptor does not have O_NONBLOCK set, the *sendmsg*( ) function shall block until
64895 space is available or a timeout occurs (see SO_SNDTIMEO in Section 2.10.16, on page 554). If
64896 space is not available at the sending socket to hold the message to be transmitted and the socket
64897 file descriptor does have O_NONBLOCK set, the *sendmsg*( ) function shall fail.

64898 If the socket protocol supports broadcast and the specified address is a broadcast address for the
64899 socket protocol, *sendmsg*( ) shall fail if the SO_BROADCAST option is not set for the socket.

64900 The socket in use may require the process to have appropriate privileges to use the *sendmsg*( )
64901 function.

64902 **RETURN VALUE**

64903 Upon successful completion, *sendmsg*( ) shall return the number of bytes sent. Otherwise, –1
64904 shall be returned and *errno* set to indicate the error.

64905 **ERRORS**

64906 The *sendmsg*( ) function shall fail if:

64907 [EAGAIN] or [EWOULDBLOCK]
64908 The socket's file descriptor is marked O_NONBLOCK and the requested
64909 operation would block. See also SO_SNDTIMEO in Section 2.10.16 (on page
64910 554).

64911 [EAFNOSUPPORT]
64912 Addresses in the specified address family cannot be used with this socket.

64913 [EBADF] The *socket* argument is not a valid file descriptor.

64914 [ECONNRESET] A connection was forcibly closed by a peer.

64915 [EINTR] A signal interrupted *sendmsg*( ) before any data was transmitted.

64916 [EINVAL] The sum of the *iov_len* values overflows an **ssize_t**.

64917 [EMSGSIZE] The message is too large to be sent all at once (as the socket requires), or the
64918 *msg_iovlen* member of the **msghdr** structure pointed to by *message* is less than
64919 or equal to 0 or is greater than {IOV_MAX}.

64920 [ENOTCONN] The socket is connection-mode but is not connected.

64921 [ENOTSOCK] The *socket* argument does not refer to a socket.

64922 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or
64923 more of the values set in *flags*.

64924 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is
64925 no longer connected. In the latter case, and if the socket is of type
64926 SOCK_STREAM or SOCK_SEQPACKET and the MSG_NOSIGNAL flag is not
64927 set, the SIGPIPE signal is generated to the calling thread.

64928 If the address family of the socket is AF_UNIX, then *sendmsg*( ) shall fail if:

64929 [EIO] An I/O error occurred while reading from or writing to the file system.

64930 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname
64931 in the socket address.

64932 [ENAMETOOLONG]
64933 The length of a component of a pathname is longer than {NAME_MAX}.

64934 [ENOENT] A component of the pathname does not name an existing file or the path name
64935 is an empty string.

64936 [ENOTDIR] A component of the path prefix of the pathname in the socket address names
64937 an existing file that is neither a directory nor a symbolic link to a directory, or
64938 the pathname in the socket address contains at least one non-<slash> character
64939 and ends with one or more trailing <slash> characters and the last pathname
64940 component names an existing file that is neither a directory nor a symbolic
64941 link to a directory.

The *sendmsg*( ) function may fail if:

[EACCES] Search permission is denied for a component of the path prefix; or write access to the named socket is denied.

[EDESTADDRREQ] The socket is not connection-mode and does not have its peer address set, and no destination address was specified.

[EHOSTUNREACH] The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).

[EIO] An I/O error occurred while reading from or writing to the file system.

[EISCONN] A destination address was specified and the socket is already connected.

[ENETDOWN] The local network interface used to reach the destination is down.

[ENETUNREACH] No route to the network is present.

[ENOBUFS] Insufficient resources were available in the system to perform the operation.

[ENOMEM] Insufficient memory was available to fulfill the request.

If the address family of the socket is AF_UNIX, then *sendmsg*( ) may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in the socket address.

[ENAMETOOLONG] The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

EXAMPLES

Done.

APPLICATION USAGE

The *select*( ) and *poll*( ) functions can be used to determine when it is possible to send more data.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getsockopt( ), *poll*( ), *pselect*( ), *recv*( ), *recvfrom*( ), *recvmsg*( ), *send*( ), *sendto*( ), *setsockopt*( ), *shutdown*( ), *socket*( )

XBD **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #073 is applied, updating the DESCRIPTION.

Austin Group Interpretation 1003.1-2001 #143 is applied.

The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The [EPIPE] error is modified.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0544 [324] is applied.

Issue 8

Austin Group Defect 1429 is applied, clarifying the behavior on timeout by adding references to Section 2.10.16 (on page 554).

64991 **NAME**

64992 sendto — send a message on a socket

64993 **SYNOPSIS**

64994 ```
#include <sys/socket.h>

ssize_t sendto(int socket, const void *message, size_t length,
    int flags, const struct sockaddr *dest_addr,
    socklen_t dest_len);
```

64998 **DESCRIPTION**

64999 The *sendto*( ) function shall send a message through a connection-mode or connectionless-mode
65000 socket.

65001 If the socket is a connectionless-mode socket, the message shall be sent to the address specified
65002 by *dest_addr* if no pre-specified peer address has been set. If a peer address has been pre-
65003 specified, either the message shall be sent to the address specified by *dest_addr* (overriding the
65004 pre-specified peer address), or the function shall return –1 and set *errno* to [EISCONN].

65005 If the socket is connection-mode, *dest_addr* shall be ignored.

65006 The *sendto*( ) function takes the following arguments:

65007 *socket* Specifies the socket file descriptor.

65008 *message* Points to a buffer containing the message to be sent.

65009 *length* Specifies the size of the message in bytes.

65010 *flags* Specifies the type of message transmission. Values of this argument are
65011 formed by logically OR'ing zero or more of the following flags:

65012 MSG_EOR Terminates a record (if supported by the protocol).

65013 MSG_OOB Sends out-of-band data on sockets that support out-of-
65014 band data. The significance and semantics of out-of-
65015 band data are protocol-specific.

65016 MSG_NOSIGNAL Requests not to send the SIGPIPE signal if an attempt to
65017 send is made on a stream-oriented socket that is no
65018 longer connected. The [EPIPE] error shall still be
65019 returned.

65020 *dest_addr* Points to a **sockaddr** structure containing the destination address. The length
65021 and format of the address depend on the address family of the socket.

65022 *dest_len* Specifies the length of the **sockaddr** structure pointed to by the *dest_addr*
65023 argument.

65024 If the address family of the socket is AF_UNIX, the application shall ensure that a null
65025 terminator after the pathname is included in the *sun_path* member of *dest_addr* as a **sockaddr_un**
65026 structure, and that *dest_len* is at least `offsetof(struct sockaddr_un, sun_path) + 1`
65027 plus the length of the pathname.

65028 If the socket protocol supports broadcast and the specified address is a broadcast address for the
65029 socket protocol, *sendto*( ) shall fail if the SO_BROADCAST option is not set for the socket.

65030 The *dest_addr* argument specifies the address of the target.

65031 The *length* argument specifies the length of the message.

65032 Successful completion of a call to *sendto*( ) does not guarantee delivery of the message. A return

  
```

value of –1 indicates only locally-detected errors.

If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does not have O_NONBLOCK set, *sendto*( ) shall block until space is available or a timeout occurs (see SO_SNDTIMEO in Section 2.10.16, on page 554). If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does have O_NONBLOCK set, *sendto*( ) shall fail.

The socket in use may require the process to have appropriate privileges to use the *sendto*( ) function.

RETURN VALUE

Upon successful completion, *sendto*( ) shall return the number of bytes sent. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *sendto*( ) function shall fail if:

[EAFNOSUPPORT]
Addresses in the specified address family cannot be used with this socket.

[EAGAIN] or [EWOULDBLOCK]
The socket's file descriptor is marked O_NONBLOCK and the requested operation would block. See also SO_SNDTIMEO in Section 2.10.16 (on page 554).

[EBADF] The *socket* argument is not a valid file descriptor.

[ECONNRESET] A connection was forcibly closed by a peer.

[EINTR] A signal interrupted *sendto*( ) before any data was transmitted.

[EMSGSIZE] The message is too large to be sent all at once, as the socket requires.

[ENOTCONN] The socket is connection-mode but is not connected.

[ENOTSOCK] The *socket* argument does not refer to a socket.

[EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or more of the values set in *flags*.

[EPIPE] The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type SOCK_STREAM or SOCK_SEQPACKET and the MSG_NOSIGNAL flag is not set, the SIGPIPE signal is generated to the calling thread.

If the address family of the socket is AF_UNIX, then *sendto*( ) shall fail if:

[EIO] An I/O error occurred while reading from or writing to the file system.

[ELOOP] A loop exists in symbolic links encountered during resolution of the pathname in the socket address.

[ENAMETOOLONG]
The length of a component of a pathname is longer than {NAME_MAX}.

[ENOENT] A component of the pathname does not name an existing file or the pathname is an empty string.

[ENOTDIR] A component of the path prefix of the pathname in the socket address names an existing file that is neither a directory nor a symbolic link to a directory, or the pathname in the socket address contains at least one non-<slash> character

and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

The *sendto*( ) function may fail if:

[EACCES] Search permission is denied for a component of the path prefix; or write access to the named socket is denied.

[EDESTADDRREQ]
The socket is not connection-mode and does not have its peer address set, and no destination address was specified.

[EHOSTUNREACH]
The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).

[EINVAL] The *dest_len* argument is not a valid length for the address family.

[EIO] An I/O error occurred while reading from or writing to the file system.

[EISCONN] A destination address was specified and the socket is already connected.

[ENETDOWN] The local network interface used to reach the destination is down.

[ENETUNREACH]
No route to the network is present.

[ENOBUFS] Insufficient resources were available in the system to perform the operation.

[ENOMEM] Insufficient memory was available to fulfill the request.

If the address family of the socket is AF_UNIX, then *sendto*( ) may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in the socket address.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

EXAMPLES

None.

APPLICATION USAGE

The *select*( ) and *poll*( ) functions can be used to determine when it is possible to send more data.

For AF_UNIX sockets, some implementations support an extension where *dest_len* does not have to include a null terminator for the pathname stored in *sun_path*, which in turn allows a pathname to be one byte longer. However, such usage is not portable, and carries a risk of accessing beyond the intended bounds of the pathname length.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getsockopt( ), *poll( )*, *pselect( )*, *recv( )*, *recvfrom( )*, *recvmsg( )*, *send( )*, *sendmsg( )*, *setsockopt( )*, *shutdown( )*, *socket( )*

XBD **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

Issue 7

Austin Group Interpretations 1003.1-2001 #035 and #073 are applied, updating the [EISCONN] error and the DESCRIPTION.

Austin Group Interpretation 1003.1-2001 #143 is applied, clarifying the [ENAMETOOLONG] error condition.

The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The [EPIPE] error is modified.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0545 [324] is applied.

Issue 8

Austin Group Defect 561 is applied, changing the requirements for the *sun_path* member of the **sockaddr_un** structure.

Austin Group Defect 1429 is applied, clarifying the behavior on timeout by adding references to Section 2.10.16 (on page 554).

65136 **NAME**

65137 setbuf — assign buffering to a stream

65138 **SYNOPSIS**

65139 ```
#include <stdio.h>
```

65140 ```
void setbuf(FILE *restrict stream, char *restrict buf);
```

65141 **DESCRIPTION**

65142 CX The functionality described on this reference page is aligned with the ISO C standard. Any
65143 conflict between the requirements described here and the ISO C standard is unintentional. This
65144 volume of POSIX.1-2024 defers to the ISO C standard.

65145 Except that it returns no value, the function call:

65146 ```
setbuf(stream, buf)
```

65147 shall be equivalent to:

65148 ```
setvbuf(stream, buf, _IOFBF, BUFSIZ)
```

65149 if *buf* is not a null pointer, or to:

65150 ```
setvbuf(stream, buf, _IONBF, BUFSIZ)
```

65151 if *buf* is a null pointer.

65152 **RETURN VALUE**

65153 The *setbuf*( ) function shall not return a value.

65154 **ERRORS**

65155 Although the *setvbuf*( ) interface may set *errno* in defined ways, the value of *errno* after a call to
65156 *setbuf*( ) is unspecified.

65157 **EXAMPLES**

65158 None.

65159 **APPLICATION USAGE**

65160 A common source of error is allocating buffer space as an ``automatic'' variable in a code block,
65161 and then failing to close the stream in the same block.

65162 With *setbuf*( ), allocating a buffer of BUFSIZ bytes does not necessarily imply that all of BUFSIZ
65163 bytes are used for the buffer area.

65164 Since *errno* is not required to be unchanged on success, in order to correctly detect and possibly
65165 recover from errors, applications should use *setvbuf*( ) instead of *setbuf*( ).

65166 **RATIONALE**

65167 None.

65168 **FUTURE DIRECTIONS**

65169 None.

65170 **SEE ALSO**

65171 Section 2.5 (on page 521), *fopen*( ), *setvbuf*( )

65172 XBD **<stdio.h>**

65173 **CHANGE HISTORY**

65174 First released in Issue 1. Derived from Issue 1 of the SVID.

  
```

Issue 6

The prototype for *setbuf*( ) is updated for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0546 [397], XSH/TC1-2008/0547 [397], and XSH/TC1-2008/0548 [14] are applied.

NAME

setegid — set the effective group ID

SYNOPSIS

```
#include <unistd.h>

int setegid(gid_t gid);
```

DESCRIPTION

If *gid* is equal to the real group ID or the saved set-group-ID, or if the process has appropriate privileges, *setegid*( ) shall set the effective group ID of the calling process to *gid*; the real group ID, saved set-group-ID, and any supplementary group IDs shall remain unchanged.

The *setegid*( ) function shall not affect the supplementary group list in any way.

RETURN VALUE

Upon successful completion, 0 shall be returned; otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *setegid*( ) function shall fail if:

[EINVAL] The value of the *gid* argument is invalid and is not supported by the implementation.

[EPERM] The process does not have appropriate privileges and *gid* does not match the real group ID or the saved set-group-ID.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to the RATIONALE section in *setuid*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getresgid*( ), *getresuid*( ), *getuid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setresgid*( ), *setresuid*( ), *setreuid*( ), *setuid*( )

XBD **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

Issue 8

Austin Group Defect 1344 is applied, adding *getresgid*( ), *getresuid*( ), *setresgid*( ), and *setresuid*( ) to SEE ALSO.

NAME

setenv — add or change environment variable

SYNOPSIS

CX
```
#include <stdlib.h>

int setenv(const char *envname, const char *envval, int overwrite);
```

DESCRIPTION

The *setenv*( ) function shall update or add a variable in the environment of the calling process. The *envname* argument points to a string containing the name of an environment variable to be added or altered. The environment variable shall be set to the value to which *envval* points. The function shall fail if *envname* points to a string which contains an '=' character. If the environment variable named by *envname* already exists and the value of *overwrite* is non-zero, the function shall return success and the environment shall be updated. If the environment variable named by *envname* already exists and the value of *overwrite* is zero, the function shall return success and the environment shall remain unchanged.

The *setenv*( ) function shall update the list of pointers to which *environ* points.

The strings described by *envname* and *envval* are copied by this function.

The *setenv*( ) function need not be thread-safe.

RETURN VALUE

Upon successful completion, zero shall be returned. Otherwise, –1 shall be returned, *errno* set to indicate the error, and the environment shall be unchanged.

ERRORS

The *setenv*( ) function shall fail if:

[EINVAL] The *envname* argument points to an empty string or points to a string containing an '=' character.

[ENOMEM] Insufficient memory was available to add a variable or its value to the environment.

EXAMPLES

None.

APPLICATION USAGE

See *exec*( ) for restrictions on changing the environment in multi-threaded applications.

RATIONALE

Unanticipated results may occur if *setenv*( ) changes the external variable *environ*. In particular, if the optional *envp* argument to *main*( ) is present, it is not changed, and thus may point to an obsolete copy of the environment (as may any other copy of *environ*). However, other than the aforementioned restriction, the standard developers intended that the traditional method of walking through the environment by way of the *environ* pointer must be supported.

It was decided that *setenv*( ) should be required by this version because it addresses a piece of missing functionality, and does not impose a significant burden on the implementor.

There was considerable debate as to whether the System V *putenv*( ) function or the BSD *setenv*( ) function should be required as a mandatory function. The *setenv*( ) function was chosen because it permitted the implementation of the *unsetenv*( ) function to delete environmental variables, without specifying an additional interface. The *putenv*( ) function is available as part of the XSI option.

The standard developers considered requiring that *setenv*( ) indicate an error when a call to it would result in exceeding {ARG_MAX}. The requirement was rejected since the condition might be temporary, with the application eventually reducing the environment size. The ultimate success or failure depends on the size at the time of a call to *exec*, which returns an indication of this error condition.

See also the RATIONALE section in *getenv*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getenv*( ), *putenv*( ), *unsetenv*( )

XBD **<stdlib.h>**, **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/55 is applied, adding references to *exec* in the APPLICATION USAGE and SEE ALSO sections.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0549 [167], XSH/TC1-2008/0550 [185], XSH/TC1-2008/0551 [167], and XSH/TC1-2008/0552 [38] are applied.

NAME

seteuid — set effective user ID

SYNOPSIS

```
#include <unistd.h>

int seteuid(uid_t uid);
```

DESCRIPTION

If *uid* is equal to the real user ID or the saved set-user-ID, or if the process has appropriate privileges, *seteuid*( ) shall set the effective user ID of the calling process to *uid*; the real user ID and saved set-user-ID shall remain unchanged.

The *seteuid*( ) function shall not affect the supplementary group list in any way.

RETURN VALUE

Upon successful completion, 0 shall be returned; otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *seteuid*( ) function shall fail if:

[EINVAL] The value of the *uid* argument is invalid and is not supported by the implementation.

[EPERM] The process does not have appropriate privileges and *uid* does not match the real user ID or the saved set-user-ID.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to the RATIONALE section in *setuid*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getresgid*( ), *getresuid*( ), *getuid*( ), *setegid*( ), *setgid*( ), *setregid*( ), *setresgid*( ), *setresuid*( ), *setreuid*( ), *setuid*( )

XBD **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/123 is applied, making an editorial correction to the [EPERM] error in the ERRORS section.

Issue 8

Austin Group Defect 1344 is applied, adding *getresgid*( ), *getresuid*( ), *setresgid*( ), and *setresuid*( ) to SEE ALSO.

65316 **NAME**

65317 setgid — set-group-ID

65318 **SYNOPSIS**

65319 `#include <unistd.h>`

65320 `int setgid(gid_t gid);`

65321 **DESCRIPTION**

65322 If the process has appropriate privileges, *setgid*( ) shall set the real group ID, effective group ID,
65323 and the saved set-group-ID of the calling process to *gid*.

65324 If the process does not have appropriate privileges, but *gid* is equal to the real group ID or the
65325 saved set-group-ID, *setgid*( ) shall set the effective group ID to *gid*; the real group ID and saved
65326 set-group-ID shall remain unchanged.

65327 The *setgid*( ) function shall not affect the supplementary group list in any way.

65328 Any supplementary group IDs of the calling process shall remain unchanged.

65329 **RETURN VALUE**

65330 Upon successful completion, 0 is returned. Otherwise, –1 shall be returned and *errno* set to
65331 indicate the error.

65332 **ERRORS**

65333 The *setgid*( ) function shall fail if:

65334 [EINVAL] The value of the *gid* argument is invalid and is not supported by the
65335 implementation.

65336 [EPERM] The process does not have appropriate privileges and *gid* does not match the
65337 real group ID or the saved set-group-ID.

65338 **EXAMPLES**

65339 None.

65340 **APPLICATION USAGE**

65341 None.

65342 **RATIONALE**

65343 Refer to the RATIONALE section in *setuid*( ).

65344 **FUTURE DIRECTIONS**

65345 None.

65346 **SEE ALSO**

65347 *exec*, *getegid*( ), *geteuid*( ), *getgid*( ), *getresgid*( ), *getresuid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setregid*( ),
65348 *setresgid*( ), *setresuid*( ), *setreuid*( ), *setuid*( )

65349 XBD **<sys/types.h>**, **<unistd.h>**

65350 **CHANGE HISTORY**

65351 First released in Issue 1. Derived from Issue 1 of the SVID.

65352 **Issue 6**

65353 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

65354 The following new requirements on POSIX implementations derive from alignment with the
65355 Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- Functionality associated with _POSIX_SAVED_IDS is now mandated. This is a FIPS requirement.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effects of *setgid*( ) in processes without appropriate privileges are changed.
- A requirement that the supplementary group list is not affected is added.

Issue 8

Austin Group Defect 1344 is applied, adding *getresgid*( ), *getresuid*( ), *setresgid*( ), and *setresuid*( ) to SEE ALSO.

NAME

setgrent — reset the group database to the first entry

SYNOPSIS

XSI
```
#include <grp.h>

void setgrent(void);
```

DESCRIPTION

Refer to *endgrent( )*.

65374 **NAME**

65375 sethostent — network host database functions

65376 **SYNOPSIS**

65377 `#include <netdb.h>`

65378 `void sethostent(int stayopen);`

65379 **DESCRIPTION**

65380 Refer to *endhostent( )*.

NAME

setjmp — set jump point for a non-local goto

SYNOPSIS

```
#include <setjmp.h>

int setjmp(jmp_buf env);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

A call to *setjmp*( ) shall save the calling environment in its *env* argument for later use by *longjmp*( ).

It is unspecified whether *setjmp*( ) is a macro or a function. If a macro definition is suppressed in order to access an actual function, or a program defines an external identifier with the name *setjmp*, the behavior is undefined.

An application shall ensure that an invocation of *setjmp*( ) appears in one of the following contexts only:

- The entire controlling expression of a selection or iteration statement
- One operand of a relational or equality operator with the other operand an integral constant expression, with the resulting expression being the entire controlling expression of a selection or iteration statement
- The operand of a unary '!' operator with the resulting expression being the entire controlling expression of a selection or iteration
- The entire expression of an expression statement (possibly cast to **void**)

If the invocation appears in any other context, the behavior is undefined.

RETURN VALUE

If the return is from a direct invocation, *setjmp*( ) shall return 0. If the return is from a call to *longjmp*( ), *setjmp*( ) shall return a non-zero value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

In general, *sigsetjmp*( ) is more useful in dealing with errors and interrupts encountered in a low-level subroutine of a program.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

longjmp( ), *sigsetjmp*( )

XBD **<setjmp.h>**

65422 **CHANGE HISTORY**

65423 First released in Issue 1. Derived from Issue 1 of the SVID.

65424 **Issue 6**

65425 The normative text is updated to avoid use of the term ``must'' for application requirements.

NAME

setkey — set encoding key (**CRYPT**)

SYNOPSIS

OB XSI
```
#include <stdlib.h>

void setkey(const char *key);
```

DESCRIPTION

The *setkey*( ) function provides access to an implementation-defined encoding algorithm. The argument of *setkey*( ) is an array of length 64 bytes containing only the bytes with numerical value of 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored; this gives a 56-bit key which is used by the algorithm. This is the key that shall be used with the algorithm to encode a string *block* passed to *encrypt*( ).

The *setkey*( ) function shall not change the setting of *errno* if successful. An application wishing to check for error situations should set *errno* to 0 before calling *setkey*( ). If *errno* is non-zero on return, an error has occurred.

The *setkey*( ) function need not be thread-safe.

RETURN VALUE

No values are returned.

ERRORS

The *setkey*( ) function shall fail if:

[ENOSYS] The functionality is not supported on this implementation.

EXAMPLES

None.

APPLICATION USAGE

Decoding need not be implemented in all environments. This is related to government restrictions in some countries on encryption and decryption routines. Historical practice has been to ship a different version of the encryption library without the decryption feature in the routines supplied. Thus the exported version of *encrypt*( ) does encoding but not decoding.

RATIONALE

None.

FUTURE DIRECTIONS

The *setkey*( ) function may be removed in a future version.

SEE ALSO

crypt( ), *encrypt*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0320 [899] is applied.

65467 **Issue 8**
65468 Austin Group Defect 1192 is applied, marking the *setkey*( ) function as obsolescent.

NAME

setlocale — set program locale

SYNOPSIS

```
#include <locale.h>

char *setlocale(int category, const char *locale);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *setlocale*( ) function selects the appropriate piece of the global locale, as specified by the *category* and *locale* arguments, and can be used to change or query the entire global locale or portions thereof. The value LC_ALL for *category* names the entire global locale; other values for *category* name only a part of the global locale:

LC_COLLATE Affects the behavior of regular expressions and the collation functions.

LC_CTYPE Affects the behavior of regular expressions, character classification, character conversion functions, and wide-character functions.

CX LC_MESSAGES Affects the affirmative and negative response expressions returned by *nl_langinfo*( ) and the way message catalogs are located. It may also affect the behavior of functions that return or write message strings.

LC_MONETARY Affects the behavior of functions that handle monetary values.

LC_NUMERIC Affects the behavior of functions that handle numeric values.

LC_TIME Affects the behavior of the time conversion functions.

The *locale* argument is a pointer to a character string containing the required setting of *category*. The contents of this string are implementation-defined. In addition, the following preset values of *locale* are defined for all settings of *category*:

CX "POSIX" Specifies the minimal environment for C-language translation called the POSIX locale. The POSIX locale is the default global locale at entry to *main*( ).

"C" Equivalent to "POSIX".

CX "" Specifies an implementation-defined native environment. The determination of the name of the new locale for the specified category depends on the value of the associated environment variables, *LC_** and *LANG*; see XBD Chapter 7 (on page 127) and Chapter 8 (on page 167).

A null pointer Directs *setlocale*( ) to query the current global locale setting and return the name of the locale if *category* is not LC_ALL, or a string which encodes the locale name(s) for all of the individual categories if *category* is LC_ALL.

CX Setting all of the categories of the global locale is similar to successively setting each individual category of the global locale, except that all error checking is done before any actions are performed. To set all the categories of the global locale, *setlocale*( ) can be invoked as:

```
setlocale(LC_ALL, "");
```

In this case, *setlocale*( ) shall first verify that the values of all the environment variables it needs according to the precedence rules (described in XBD Chapter 8, on page 167) indicate supported locales. If the value of any of these environment variable searches yields a locale that is not supported (and non-null), *setlocale*( ) shall return a null pointer and the global locale shall not be

changed. If all environment variables name supported locales, *setlocale*( ) shall proceed as if it had been called for each category, using the appropriate value from the associated environment variable or from the implementation-defined default if there is no such value.

The global locale established using *setlocale*( ) shall only be used in threads for which no current locale has been set using *uselocale*( ) or whose current locale has been set to the global locale using *uselocale*(LC_GLOBAL_LOCALE).

The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls *setlocale*( ).

The *setlocale*( ) function need not be thread-safe; however, it shall avoid data races with all function calls that do not affect and are not affected by the global locale.

RETURN VALUE

Upon successful completion, *setlocale*( ) shall return the string associated with the specified category for the new locale. Otherwise, *setlocale*( ) shall return a null pointer and the global locale shall not be changed.

A null pointer for *locale* shall cause *setlocale*( ) to return a pointer to the string associated with the specified *category* for the current global locale. The global locale shall not be changed.

The string returned by *setlocale*( ) is such that a subsequent call with that string and its associated *category* shall restore that part of the global locale. The application shall not modify the string returned. CX The returned string pointer might be invalidated or the string content might be overwritten by a subsequent call to *setlocale*( ). CX The returned pointer might also be invalidated if the calling thread is terminated.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The following code illustrates how a program can initialize the international environment for one language, while selectively modifying the global locale such that regular expressions and string operations can be applied to text recorded in a different language:

```
setlocale(LC_ALL, "De");
setlocale(LC_COLLATE, "Fr@dict");
```

Internationalized programs can initiate language operation according to environment variable settings (see XBD Section 8.2, on page 169) by calling *setlocale*( ) as follows:

```
setlocale(LC_ALL, "");
```

Changing the setting of *LC_MESSAGES* has no effect on catalogs that have already been opened by calls to *catopen*( ).

In order to make use of different locale settings while multiple threads are running, applications should use *uselocale*( ) in preference to *setlocale*( ).

RATIONALE

References to the international environment or locale in the following text relate to the global locale for the process. This can be overridden for individual threads using *uselocale*( ).

The ISO C standard defines a collection of functions to support internationalization. One of the most significant aspects of these functions is a facility to set and query the *international environment*. The international environment is a repository of information that affects the

behavior of certain functionality, namely:

1. Character handling
2. Collating
3. Date/time formatting
4. Numeric editing
5. Monetary formatting
6. Messaging

The *setlocale*( ) function provides the application developer with the ability to set all or portions, called *categories*, of the international environment. These categories correspond to the areas of functionality mentioned above. The syntax for *setlocale*( ) is as follows:

```
char *setlocale(int category, const char *locale);
```

where *category* is the name of one of following categories, namely:

LC_COLLATE
LC_CTYPE
LC_MESSAGES
LC_MONETARY
LC_NUMERIC
LC_TIME

In addition, a special value called LC_ALL directs *setlocale*( ) to set all categories.

There are two primary uses of *setlocale*( ):

1. Querying the international environment to find out what it is set to
2. Setting the international environment, or *locale*, to a specific value

The behavior of *setlocale*( ) in these two areas is described below. Since it is difficult to describe the behavior in words, examples are used to illustrate the behavior of specific uses.

To query the international environment, *setlocale*( ) is invoked with a specific category and the null pointer as the locale. The null pointer is a special directive to *setlocale*( ) that tells it to query rather than set the international environment. The following syntax is used to query the name of the international environment:

```
setlocale({LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, \
    LC_NUMERIC, LC_TIME},(char *) NULL);
```

The *setlocale*( ) function shall return the string corresponding to the current international environment. This value may be used by a subsequent call to *setlocale*( ) to reset the international environment to this value. However, it should be noted that the return value from *setlocale*( ) may be a pointer to a static area within the function and is not guaranteed to remain unchanged (that is, it may be modified by a subsequent call to *setlocale*( )). Therefore, if the purpose of calling *setlocale*( ) is to save the value of the current international environment so it can be changed and reset later, the return value should be copied to an array of **char** in the calling program.

There are three ways to set the international environment with *setlocale*( ):

setlocale(*category*, *string*)

This usage sets a specific *category* in the international environment to a specific value corresponding to the value of the *string*. A specific example is provided below:

```
setlocale(LC_ALL, "fr_FR.ISO-8859-1");
```

In this example, all categories of the international environment are set to the locale corresponding to the string `"fr_FR.ISO-8859-1"`, or to the French language as spoken in France using the ISO/IEC 8859-1: 1998 standard codeset.

If the string does not correspond to a valid locale, *setlocale*( ) shall return a null pointer and the international environment is not changed. Otherwise, *setlocale*( ) shall return the name of the locale just set.

setlocale(*category*, "C")

The ISO C standard states that one locale must exist on all conforming implementations. The name of the locale is C and corresponds to a minimal international environment needed to support the C programming language.

setlocale(*category*, "")

This sets a specific category to an implementation-defined default. This corresponds to the value of the environment variables.

FUTURE DIRECTIONS

None.

SEE ALSO

catopen( ), *exec* , *fprintf*( ), *fscanf*( ), *getlocalename_l*( ), *isalnum*( ), *isalpha*( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *iswalnum*( ), *iswalpha*( ), *iswblank*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *isxdigit*( ), *localeconv*( ), *mblen*( ), *mbstowcs*( ), *mbtowc*( ), *newlocale*( ), *nl_langinfo*( ), *perror*( ), *psiginfo*( ), *strcoll*( ), *strerror*( ), *strfmon*( ), *strsignal*( ), *strtod*( ), *strxfrm*( ), *tolower*( ), *toupper*( ), *towlower*( ), *towupper*( ), *uselocale*( ), *wcscoll*( ), *wcstod*( ), *wcstombs*( ), *wcsxfrm*( ), *wctomb*( )

XBD Chapter 7 (on page 127), Chapter 8 (on page 167), **<langinfo.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue 3.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

Extensions beyond the ISO C standard are marked.

The normative text is updated to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/124 is applied, updating the DESCRIPTION to clarify the behavior of:

```
setlocale(LC_ALL, "");
```

Issue 7

Functionality relating to the Threads option is moved to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0553 [302], XSH/TC1-2008/0554 [303], XSH/TC1-2008/0555 [302], XSH/TC1-2008/0556 [302], XSH/TC1-2008/0557 [302], XSH/TC1-2008/0558 [302], XSH/TC1-2008/0559 [302], XSH/TC1-2008/0560 [288], XSH/TC1-2008/0561 [302], XSH/TC1-2008/0562 [302], XSH/TC1-2008/0563 [302],

XSH/TC1-2008/0564 [302], XSH/TC1-2008/0565 [302], XSH/TC1-2008/0566 [302], XSH/TC1-2008/0567 [288], XSH/TC1-2008/0568 [288], and XSH/TC1-2008/0569 [303] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0321 [826], XSH/TC2-2008/0322 [826], and XSH/TC2-2008/0323 [596] are applied.

Issue 8

Austin Group Defect 1220 is applied, adding *getlocalename_l*( ) to the SEE ALSO section.

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899: 2018 standard.

NAME

setlogmask — set the log priority mask

SYNOPSIS

XSI

```
#include <syslog.h>

int setlogmask(int maskpri);
```

DESCRIPTION

Refer to *closelog( )*.

65655 **NAME**

65656 setnetent — network database function

65657 **SYNOPSIS**

65658 `#include <netdb.h>`

65659 `void setnetent(int stayopen);`

65660 **DESCRIPTION**

65661 Refer to *endnetent( )*.

65662 **NAME**

65663 setpgid — set process group ID for job control

65664 **SYNOPSIS**

65665 `#include <unistd.h>`

65666 `int setpgid(pid_t pid, pid_t pgid);`

65667 **DESCRIPTION**

65668 The *setpgid*( ) function shall either join an existing process group or create a new process group
65669 within the session of the calling process.

65670 The process group ID of a session leader shall not change.

65671 Upon successful completion, the process group ID of the process with a process ID that matches
65672 *pid* shall be set to *pgid*.

65673 As a special case, if *pid* is 0, the process ID of the calling process shall be used. Also, if *pgid* is 0,
65674 the process ID of the indicated process shall be used.

65675 **RETURN VALUE**

65676 Upon successful completion, *setpgid*( ) shall return 0; otherwise, −1 shall be returned and *errno*
65677 shall be set to indicate the error.

65678 **ERRORS**

65679 The *setpgid*( ) function shall fail if:

65680 [EACCES] The value of the *pid* argument matches the process ID of a child process of the
65681 calling process and the child process has successfully executed one of the *exec*
65682 functions.

65683 [EINVAL] The value of the *pgid* argument is less than 0, or is not a value supported by
65684 the implementation.

65685 [EPERM] The process indicated by the *pid* argument is a session leader.

65686 [EPERM] The value of the *pid* argument matches the process ID of a child process of the
65687 calling process and the child process is not in the same session as the calling
65688 process.

65689 [EPERM] The value of the *pgid* argument is valid but does not match the process ID of
65690 the process indicated by the *pid* argument and there is no process with a
65691 process group ID that matches the value of the *pgid* argument in the same
65692 session as the calling process.

65693 [ESRCH] The value of the *pid* argument does not match the process ID of the calling
65694 process or of a child process of the calling process.

65695 **EXAMPLES**

65696 None.

65697 **APPLICATION USAGE**

65698 None.

65699 **RATIONALE**

65700 The *setpgid*( ) function shall group processes together for the purpose of signaling, placement in
65701 foreground or background, and other job control actions.

65702 The *setpgid*( ) function is similar to the *setpgrp*( ) function of 4.2 BSD, except that 4.2 BSD allowed
65703 the specified new process group to assume any value. This presents certain security problems
65704 and is more flexible than necessary to support job control.

To provide tighter security, *setpgid*( ) only allows the calling process to join a process group already in use inside its session or create a new process group whose process group ID was equal to its process ID.

When a job control shell spawns a new job, the processes in the job must be placed into a new process group via *setpgid*( ). There are two timing constraints involved in this action:

1. The new process must be placed in the new process group before the appropriate program is launched via one of the *exec* functions.
2. The new process must be placed in the new process group before the shell can correctly send signals to the new process group.

To address these constraints, the following actions are performed. The new processes call *setpgid*( ) to alter their own process groups after *fork*( ) but before *exec*. This satisfies the first constraint. Under 4.3 BSD, the second constraint is satisfied by the synchronization property of *vfork*( ); that is, the shell is suspended until the child has completed the *exec*, thus ensuring that the child has completed the *setpgid*( ). A new version of *fork*( ) with this same synchronization property was considered, but it was decided instead to merely allow the parent shell process to adjust the process group of its child processes via *setpgid*( ). Both timing constraints are now satisfied by having both the parent shell and the child attempt to adjust the process group of the child process; it does not matter which succeeds first.

Since it would be confusing to an application to have its process group change after it began executing (that is, after *exec*), and because the child process would already have adjusted its process group before this, the [EACCES] error was added to disallow this.

One non-obvious use of *setpgid*( ) is to allow a job control shell to return itself to its original process group (the one in effect when the job control shell was executed). A job control shell does this before returning control back to its parent when it is terminating or suspending itself as a way of restoring its job control ``state'' back to what its parent would expect. (Note that the original process group of the job control shell typically matches the process group of its parent, but this is not necessarily always the case.)

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getpgrp*( ), *setsid*( ), *tcsetpgrp*( )

XBD **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The *setpgid*( ) function is mandatory since _POSIX_JOB_CONTROL is required to be defined in this version. This is a FIPS requirement.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/56 is applied, changing the wording in the DESCRIPTION from ``the process group ID of the indicated process shall be used'' to ``the process ID of the indicated process shall be used''. This change reverts the wording to as in the ISO POSIX-1: 1996 standard; it appeared to be an unintentional change.

65752 **NAME**

65753 setpriority — set the nice value

65754 **SYNOPSIS**

65755 XSI
```
#include <sys/resource.h>

int setpriority(int which, id_t who, int nice);
```

65757 **DESCRIPTION**

65758 Refer to *getpriority( )*.

65759 **NAME**

65760 setprotoent — network protocol database functions

65761 **SYNOPSIS**

65762 `#include <netdb.h>`

65763 `void setprotoent(int` *`stayopen`*`);`

65764 **DESCRIPTION**

65765 Refer to *endprotoent( )*.

65766 **NAME**

65767 setpwent — user database function

65768 **SYNOPSIS**

65769 XSI
```
#include <pwd.h>

void setpwent(void);
```
65770

65771 **DESCRIPTION**

65772 Refer to *endpwent( )*.

NAME

setregid — set real and effective group IDs

SYNOPSIS

XSI
```
#include <unistd.h>

int setregid(gid_t rgid, gid_t egid);
```

DESCRIPTION

The *setregid*( ) function shall set the real and effective group IDs of the calling process.

If *rgid* is –1, the real group ID shall not be changed; if *egid* is –1, the effective group ID shall not be changed.

The real and effective group IDs may be set to different values in the same call.

Only a process with appropriate privileges can set the real group ID and the effective group ID to any valid value.

A non-privileged process can set either the real group ID to the saved set-group-ID from one of the *exec* family of functions, or the effective group ID to the saved set-group-ID or the real group ID.

If the real group ID is being set (*rgid* is not –1), or the effective group ID is being set to a value not equal to the real group ID, then the saved set-group-ID of the current process shall be set equal to the new effective group ID.

Any supplementary group IDs of the calling process remain unchanged.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error, and neither of the group IDs are changed.

ERRORS

The *setregid*( ) function shall fail if:

[EINVAL] The value of the *rgid* or *egid* argument is invalid or out-of-range.

[EPERM] The process does not have appropriate privileges and a change other than changing the real group ID to the saved set-group-ID, or changing the effective group ID to the real group ID or the saved set-group-ID, was requested.

EXAMPLES

None.

APPLICATION USAGE

If a non-privileged set-group-ID process sets its effective group ID to its real group ID, it can only set its effective group ID back to the previous value if *rgid* was –1 in the *setregid*( ) call, since the saved-group-ID is not changed in that case. If *rgid* was equal to the real group ID in the *setregid*( ) call, then the saved set-group-ID will also have been changed to the real user ID.

RATIONALE

Earlier versions of this standard did not specify whether the saved set-group-ID was affected by *setregid*( ) calls. This version specifies common existing practice that constitutes an important security feature. The ability to set both the effective group ID and saved set-group-ID to be the same as the real group ID means that any security weakness in code that is executed after that point cannot result in malicious code being executed with the previous effective group ID. Privileged applications could already do this using just *setgid*( ), but for non-privileged

65816 applications the only standard method available is to use this feature of *setregid*( ).

65817 **FUTURE DIRECTIONS**

65818 None.

65819 **SEE ALSO**

65820 *exec*, *getegid*( ), *geteuid*( ), *getgid*( ), *getresgid*( ), *getresuid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ),
65821 *setresgid*( ), *setresuid*( ), *setreuid*( ), *setuid*( )

65822 XBD **<unistd.h>**

65823 **CHANGE HISTORY**

65824 First released in Issue 4, Version 2.

65825 **Issue 5**

65826 Moved from X/OPEN UNIX extension to BASE.

65827 The DESCRIPTION is updated to indicate that the saved set-group-ID can be set by any of the
65828 *exec* family of functions, not just *execve*( ).

65829 **Issue 7**

65830 SD5-XSH-ERN-177 is applied, adding the ability to set both the effective group ID and saved set-
65831 group-ID to be the same as the real group ID.

65832 **Issue 8**

65833 Austin Group Defect 1344 is applied, adding *getresgid*( ), *getresuid*( ), *setresgid*( ), and *setresuid*( ) to
65834 SEE ALSO.

NAME

setresgid — set real group ID, effective group ID, and saved set-group-ID

SYNOPSIS

XSI
```
#include <unistd.h>

int setresgid(gid_t rgid, gid_t egid, gid_t sgid);
```

DESCRIPTION

The *setresgid*( ) function shall set the real group ID, effective group ID, and saved set-group-ID of the calling process to the values specified by *rgid*, *egid*, and *sgid*, respectively.

If an argument is –1, the corresponding ID shall not be changed.

Only a process with appropriate privileges can set the real group ID, effective group ID, and saved set-group-ID to any valid value.

A non-privileged process can set its real group ID, effective group ID, and saved set-group-ID, each to one of the values that it currently holds in its real group ID, effective group ID, or saved set-group-ID.

The real group ID, effective group ID, and saved set-group-ID can be set to different values in the same call.

Any supplementary group IDs of the calling process shall remain unchanged.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error, and none of the IDs shall be changed.

ERRORS

The *setresgid*( ) function shall fail if:

[EINVAL] The value of the *rgid*, *egid*, or *sgid* argument is invalid or out-of-range.

[EPERM] The calling process does not have appropriate privileges and an attempt was made to change the real group ID, effective group ID, or saved set-group-ID to a value that is not currently present in one of those IDs.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getresgid*( ), *getresuid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setresuid*( ), *setreuid*( ), *setuid*( )

XBD **<unistd.h>**

65873 **CHANGE HISTORY**
65874 First released in Issue 8.

NAME

setresuid — set real user ID, effective user ID, and saved set-user-ID

SYNOPSIS

XSI
```
#include <unistd.h>

int setresuid(uid_t ruid, uid_t euid, uid_t suid);
```

DESCRIPTION

The *setresuid*( ) function shall set the real user ID, effective user ID, and saved set-user-ID of the calling process to the values specified by *ruid*, *euid*, and *suid*, respectively.

If an argument is –1, the corresponding ID shall not be changed.

Only a process with appropriate privileges can set the real user ID, effective user ID, and saved set-user-ID to any valid value.

A non-privileged process can set its real user ID, effective user ID, and saved set-user-ID, each to one of the values that it currently holds in its real user ID, effective user ID, or saved set-user-ID.

The real user ID, effective user ID, and saved set-user-ID can be set to different values in the same call.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error, and none of the IDs shall be changed.

ERRORS

The *setresuid*( ) function shall fail if:

[EINVAL] The value of the *ruid*, *euid*, or *suid* argument is invalid or out-of-range.

[EPERM] The calling process does not have appropriate privileges and an attempt was made to change the real user ID, effective user ID, or saved set-user-ID to a value that is not currently present in one of those IDs or an attempt was made to change the real user ID to a value not permitted by the implementation.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getresgid*( ), *getresuid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setresgid*( ), *setreuid*( ), *setuid*( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 8.

NAME

setreuid — set real and effective user IDs

SYNOPSIS

XSI
```
#include <unistd.h>

int setreuid(uid_t ruid, uid_t euid);
```

DESCRIPTION

The *setreuid*( ) function shall set the real and effective user IDs of the current process to the values specified by the *ruid* and *euid* arguments. If *ruid* or *euid* is –1, the corresponding effective or real user ID of the current process shall be left unchanged.

A process with appropriate privileges can set either ID to any value. An unprivileged process can only set the effective user ID if the *euid* argument is equal to either the real, effective, or saved user ID of the process.

If the real user ID is being set (*ruid* is not –1), or the effective user ID is being set to a value not equal to the real user ID, then the saved set-user-ID of the current process shall be set equal to the new effective user ID.

It is unspecified whether a process without appropriate privileges is permitted to change the real user ID to match the current effective user ID or saved set-user-ID of the process.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *setreuid*( ) function shall fail if:

[EINVAL] The value of the *ruid* or *euid* argument is invalid or out-of-range.

[EPERM] The current process does not have appropriate privileges, and either an attempt was made to change the effective user ID to a value other than the real user ID or the saved set-user-ID or an attempt was made to change the real user ID to a value not permitted by the implementation.

EXAMPLES

Setting the Effective User ID to the Real User ID

The following example sets the effective user ID of the calling process to the real user ID, so that files created later will be owned by the current user. It also sets the saved set-user-ID to the real user ID, so any future attempt to set the effective user ID back to its previous value will fail.

```
#include <unistd.h>
#include <sys/types.h>
...
setreuid(getuid(), getuid());
...
```

APPLICATION USAGE

None.

RATIONALE

Earlier versions of this standard did not specify whether the saved set-user-ID was affected by *setreuid*( ) calls. This version specifies common existing practice that constitutes an important security feature. The ability to set both the effective user ID and saved set-user-ID to be the same as the real user ID means that any security weakness in code that is executed after that point cannot result in malicious code being executed with the previous effective user ID. Privileged applications could already do this using just *setuid*( ), but for non-privileged applications the only standard method available is to use this feature of *setreuid*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

getegid( ), *geteuid*( ), *getgid*( ), *getresgid*( ), *getresuid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setresgid*( ), *setresuid*( ), *setuid*( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 7

SD5-XSH-ERN-177 is applied, adding the ability to set both the effective user ID and the saved set-user-ID to be the same as the real user ID.

Issue 8

Austin Group Defect 1344 is applied, adding *getresgid*( ), *getresuid*( ), *setresgid*( ), and *setresuid*( ) to SEE ALSO.

65977 **NAME**

65978 setrlimit — control maximum resource consumption

65979 **SYNOPSIS**

65980 `#include <sys/resource.h>`

65981 `int setrlimit(int resource, const struct rlimit *rlp);`

65982 **DESCRIPTION**

65983 Refer to *getrlimit( )*.

65984 **NAME**

65985 setservent — network services database functions

65986 **SYNOPSIS**

65987 `#include <netdb.h>`

65988 `void setservent(int` *`stayopen`*`);`

65989 **DESCRIPTION**

65990 Refer to *endservent( )*.

NAME

setsid — create session and set process group ID

SYNOPSIS

```
#include <unistd.h>

pid_t setsid(void);
```

DESCRIPTION

The *setsid*( ) function shall create a new session, if the calling process is not a process group leader. Upon return the calling process shall be the session leader of this new session, shall be the process group leader of a new process group, and shall have no controlling terminal. The process group ID of the calling process shall be set equal to the process ID of the calling process. The calling process shall be the only process in the new process group and the only process in the new session.

RETURN VALUE

Upon successful completion, *setsid*( ) shall return the value of the new process group ID of the calling process. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *setsid*( ) function shall fail if:

[EPERM] The calling process is already a process group leader, or the process group ID of a process other than the calling process matches the process ID of the calling process.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *setsid*( ) function is similar to the *setpgrp*( ) function of System V. System V, without job control, groups processes into process groups and creates new process groups via *setpgrp*( ); only one process group may be part of a login session.

Job control allows multiple process groups within a login session. In order to limit job control actions so that they can only affect processes in the same login session, this volume of POSIX.1-2024 adds the concept of a session that is created via *setsid*( ). The *setsid*( ) function also creates the initial process group contained in the session. Additional process groups can be created via the *setpgid*( ) function. A System V process group would correspond to a POSIX System Interfaces session containing a single POSIX process group. Note that this function requires that the calling process not be a process group leader. The usual way to ensure this is true is to create a new process with *fork*( ) and have it call *setsid*( ). The *fork*( ) function guarantees that the process ID of the new process does not match any existing process group ID.

FUTURE DIRECTIONS

None.

SEE ALSO

getsid( ), *setpgid*( )

XBD **<sys/types.h>**, **<unistd.h>**

66033 **CHANGE HISTORY**

66034 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

66035 **Issue 6**

66036 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

66037 The following new requirements on POSIX implementations derive from alignment with the
66038 Single UNIX Specification:

66039 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
66040 required for conforming implementations of previous POSIX specifications, it was not
66041 required for UNIX applications.

66042 **Issue 7**

66043 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0570 [421] is applied.

66044 **Issue 8**

66045 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

66046 **NAME**

66047 setsockopt — set the socket options

66048 **SYNOPSIS**

66049 ```
#include <sys/socket.h>
```

66050 ```
int setsockopt(int socket, int level, int option_name,
```

66051 ```
    const void *option_value, socklen_t option_len);
```

66052 **DESCRIPTION**

66053 The *setsockopt*( ) function shall set the option specified by the *option_name* argument, at the
66054 protocol level specified by the *level* argument, to the value pointed to by the *option_value*
66055 argument for the socket associated with the file descriptor specified by the *socket* argument.

66056 The *level* argument specifies the protocol level at which the option resides. To set options at the
66057 socket level, specify the *level* argument as SOL_SOCKET. To set options at other levels, supply
66058 the appropriate *level* identifier for the protocol controlling the option. For example, to indicate
66059 that an option is interpreted by the TCP (Transport Control Protocol), set *level* to IPPROTO_TCP
66060 as defined in the **<netinet/in.h>** header.

66061 The *option_name* argument specifies a single option to set. It can be one of the socket-level
66062 options defined in **<sys/socket.h>** and described in Section 2.10.16 (on page 554). If *option_name*
66063 is equal to SO_RCVTIMEO or SO_SNDTIMEO and the implementation supports setting the
66064 option, it is unspecified whether the **struct timeval** pointed to by *option_value* is stored as
66065 provided by this function or is rounded up to align with the resolution of the clock being used. If
66066 *setsockopt*( ) is called with *option_name* equal to SO_ACCEPTCONN, SO_ERROR, or SO_TYPE,
66067 the behavior is unspecified.

66068 **RETURN VALUE**

66069 Upon successful completion, *setsockopt*( ) shall return 0. Otherwise, –1 shall be returned and
66070 *errno* set to indicate the error.

66071 **ERRORS**

66072 The *setsockopt*( ) function shall fail if:

66073 [EBADF] The *socket* argument is not a valid file descriptor.

66074 [EDOM] The send and receive timeout values are too big to fit into the timeout fields in
66075 the socket structure.

66076 [EINVAL] The specified option is invalid at the specified socket level or the socket has
66077 been shut down.

66078 [EISCONN] The socket is already connected, and a specified option cannot be set while the
66079 socket is connected.

66080 [ENOPROTOOPT]
66081 The option is not supported by the protocol.

66082 [ENOTSOCK] The *socket* argument does not refer to a socket.

66083 The *setsockopt*( ) function may fail if:

66084 [ENOMEM] There was insufficient memory available for the operation to complete.

66085 [ENOBUFS] Insufficient resources are available in the system to complete the call.

  
```

EXAMPLES

None.

APPLICATION USAGE

The *setsockopt*( ) function provides an application program with the means to control socket behavior. An application program can use *setsockopt*( ) to allocate buffer space, control timeouts, or permit socket data broadcasts. The **<sys/socket.h>** header defines the socket-level options available to *setsockopt*( ).

Options may exist at multiple protocol levels. The SO_ options are always present at the uppermost socket level.

It is implementation-defined which socket options, if any, are inherited from a listening socket to an accepted socket by *accept*( ) or *accept4*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.10 (on page 549), *bind*( ), *endprotoent*( ), *getsockopt*( ), *socket*( )

XBD **<netinet/in.h>**, **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/125 is applied, updating the SO_LINGER option in the DESCRIPTION to refer to the calling thread rather than the process.

Issue 7

Austin Group Interpretation 1003.1-2001 #158 is applied, removing text relating to socket options that is now in Section 2.10.16 (on page 554).

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0571 [369] is applied.

Issue 8

Austin Group Defect 1337 is applied, clarifying socket option default values.

66114 **NAME**

66115 setstate — switch pseudo-random number generator state arrays

66116 **SYNOPSIS**

66117 XSI
```
#include <stdlib.h>

char *setstate(char *state);
```
66118

66119 **DESCRIPTION**

66120 Refer to *initstate*( ).

66121 **NAME**

66122 setuid — set user ID

66123 **SYNOPSIS**

66124 `#include <unistd.h>`

66125 `int setuid(uid_t uid);`

66126 **DESCRIPTION**

66127 If the process has appropriate privileges, *setuid*( ) shall set the real user ID, effective user ID, and
66128 the saved set-user-ID of the calling process to *uid*.

66129 If the process does not have appropriate privileges, but *uid* is equal to the real user ID or the
66130 saved set-user-ID, *setuid*( ) shall set the effective user ID to *uid*; the real user ID and saved set-
66131 user-ID shall remain unchanged.

66132 The *setuid*( ) function shall not affect the supplementary group list in any way.

66133 **RETURN VALUE**

66134 Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to
66135 indicate the error.

66136 **ERRORS**

66137 The *setuid*( ) function shall fail, return –1, and set *errno* to the corresponding value if one or more
66138 of the following are true:

66139 [EINVAL] The value of the *uid* argument is invalid and not supported by the
66140 implementation.

66141 [EPERM] The process does not have appropriate privileges and *uid* does not match the
66142 real user ID or the saved set-user-ID.

66143 **EXAMPLES**

66144 None.

66145 **APPLICATION USAGE**

66146 None.

66147 **RATIONALE**

66148 The various behaviors of the *setuid*( ) and *setgid*( ) functions when called by non-privileged
66149 processes reflect the behavior of different historical implementations. For portability, it is
66150 recommended that new non-privileged applications use the *seteuid*( ) and *setegid*( ) functions
66151 instead.

66152 The saved set-user-ID capability allows a program to regain the effective user ID established at
66153 the last *exec* call. Similarly, the saved set-group-ID capability allows a program to regain the
66154 effective group ID established at the last *exec* call. These capabilities are derived from System V.
66155 Without them, a program might have to run as superuser in order to perform the same
66156 functions, because superuser can write on the user's files. This is a problem because such a
66157 program can write on any user's files, and so must be carefully written to emulate the
66158 permissions of the calling process properly. In System V, these capabilities have traditionally
66159 been implemented only via the *setuid*( ) and *setgid*( ) functions for non-privileged processes. The
66160 fact that the behavior of those functions was different for privileged processes made them
66161 difficult to use. The POSIX.1-1990 standard defined the *setuid*( ) function to behave differently
66162 for privileged and unprivileged users. When the caller had appropriate privileges, the function
66163 set the real user ID, effective user ID, and saved set-user ID of the calling process on
66164 implementations that supported it. When the caller did not have appropriate privileges, the
66165 function set only the effective user ID, subject to permission checks. The former use is generally
66166 needed for utilities like *login* and *su*, which are not conforming applications and thus outside the

scope of POSIX.1-2024. These utilities wish to change the user ID irrevocably to a new value, generally that of an unprivileged user. The latter use is needed for conforming applications that are installed with the set-user-ID bit and need to perform operations using the real user ID.

POSIX.1-2024 augments the latter functionality with a mandatory feature named _POSIX_SAVED_IDS. This feature permits a set-user-ID application to switch its effective user ID back and forth between the values of its *exec*-time real user ID and effective user ID. Unfortunately, the POSIX.1-1990 standard did not permit a conforming application using this feature to work properly when it happened to be executed with (implementation-defined) appropriate privileges. Furthermore, the application did not even have a means to tell whether it had this privilege. Since the saved set-user-ID feature is quite desirable for applications, as evidenced by the fact that NIST required it in FIPS 151-2, it has been mandated by POSIX.1-2024. However, there are implementors who have been reluctant to support it given the limitation described above.

The 4.3BSD system handles the problem by supporting separate functions: *setuid*( ) (which always sets both the real and effective user IDs, like *setuid*( ) in POSIX.1-2024 for privileged users), and *seteuid*( ) (which always sets just the effective user ID, like *setuid*( ) in POSIX.1-2024 for non-privileged users). This separation of functionality into distinct functions seems desirable. 4.3BSD does not support the saved set-user-ID feature. It supports similar functionality of switching the effective user ID back and forth via *setreuid*( ), which permits reversing the real and effective user IDs. This model seems less desirable than the saved set-user-ID because the real user ID changes as a side-effect. The current 4.4BSD includes saved effective IDs and uses them for *seteuid*( ) and *setegid*( ) as described above. The *setreuid*( ) and *setregid*( ) functions will be deprecated or removed.

The solution here is:

- Require that all implementations support the functionality of the saved set-user-ID, which is set by the *exec* functions and by privileged calls to *setuid*( ).
- Add the *seteuid*( ) and *setegid*( ) functions as portable alternatives to *setuid*( ) and *setgid*( ) for non-privileged and privileged processes.

Historical systems have provided two mechanisms for a set-user-ID process to change its effective user ID to be the same as its real user ID in such a way that it could return to the original effective user ID: the use of the *setuid*( ) function in the presence of a saved set-user-ID, or the use of the BSD *setreuid*( ) function, which was able to swap the real and effective user IDs. The changes included in POSIX.1-2024 provide a new mechanism using *seteuid*( ) in conjunction with a saved set-user-ID. Thus, all implementations with the new *seteuid*( ) mechanism will have a saved set-user-ID for each process, and most of the behavior controlled by _POSIX_SAVED_IDS has been changed to agree with the case where the option was defined. The *kill*( ) function is an exception. Implementors of the new *seteuid*( ) mechanism will generally be required to maintain compatibility with the older mechanisms previously supported by their systems. However, compatibility with this use of *setreuid*( ) and with the _POSIX_SAVED_IDS behavior of *kill*( ) is unfortunately complicated. If an implementation with a saved set-user-ID allows a process to use *setreuid*( ) to swap its real and effective user IDs, but were to leave the saved set-user-ID unmodified, the process would then have an effective user ID equal to the original real user ID, and both real and saved set-user-ID would be equal to the original effective user ID. In that state, the real user would be unable to kill the process, even though the effective user ID of the process matches that of the real user, if the *kill*( ) behavior of _POSIX_SAVED_IDS was used. This is obviously not acceptable. The alternative choice, which is used in at least one implementation, is to change the saved set-user-ID to the effective user ID during most calls to *setreuid*( ). The standard developers considered that alternative to be less correct than the retention of the old behavior of *kill*( ) in such systems. Current conforming applications shall

accommodate either behavior from *kill*( ), and there appears to be no strong reason for *kill*( ) to check the saved set-user-ID rather than the effective user ID.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getresgid*( ), *getresuid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setresgid*( ), *setresuid*( ), *setreuid*( )

XBD **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The functionality associated with _POSIX_SAVED_IDS is now mandatory. This is a FIPS requirement.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effects of *setuid*( ) in processes without appropriate privileges are changed.
- A requirement that the supplementary group list is not affected is added.

Issue 8

Austin Group Defect 1344 is applied, adding *getresgid*( ), *getresuid*( ), *setresgid*( ), and *setresuid*( ) to SEE ALSO.

66241 **NAME**

66242 setutxent — reset the user accounting database to the first entry

66243 **SYNOPSIS**

66244 XSI
```
#include <utmpx.h>

void setutxent(void);
```

66246 **DESCRIPTION**

66247 Refer to *endutxent( )*.

66248 **NAME**

66249 setvbuf — assign buffering to a stream

66250 **SYNOPSIS**

66251
```
#include <stdio.h>
```

66252
```
int setvbuf(FILE *restrict stream, char *restrict buf, int type,
```
66253
```
    size_t size);
```

66254 **DESCRIPTION**

66255 CX The functionality described on this reference page is aligned with the ISO C standard. Any
66256 conflict between the requirements described here and the ISO C standard is unintentional. This
66257 volume of POSIX.1-2024 defers to the ISO C standard.

66258 The *setvbuf*( ) function may be used after the stream pointed to by *stream* is associated with an
66259 open file but before any other operation (other than an unsuccessful call to *setvbuf*( )) is
66260 performed on the stream. The argument *type* determines how *stream* shall be buffered, as
66261 follows:

66262 • {_IOFBF} shall cause input/output to be fully buffered.

66263 • {_IOLBF} shall cause input/output to be line buffered.

66264 • {_IONBF} shall cause input/output to be unbuffered.

66265 If *buf* is not a null pointer, the array it points to may be used instead of a buffer allocated by
66266 *setvbuf*( ) and the argument *size* specifies the size of the array; otherwise, *size* may determine the
66267 size of a buffer allocated by the *setvbuf*( ) function. The contents of the array at any time are
66268 unspecified.

66269 For information about streams, see Section 2.5 (on page 521).

66270 **RETURN VALUE**

66271 Upon successful completion, *setvbuf*( ) shall return 0. Otherwise, it shall return a non-zero value
66272 CX if an invalid value is given for *type* or if the request cannot be honored, and may set *errno* to
66273 indicate the error.

66274 **ERRORS**

66275 The *setvbuf*( ) function may fail if:

66276 CX [EBADF] The stream is not a memory stream and the file descriptor underlying the
66277 stream is not valid.

66278 **EXAMPLES**

66279 None.

66280 **APPLICATION USAGE**

66281 A common source of error is allocating buffer space as an ``automatic'' variable in a code block,
66282 and then failing to close the stream in the same block.

66283 With *setvbuf*( ), allocating a buffer of *size* bytes does not necessarily imply that all of *size* bytes are
66284 used for the buffer area.

66285 Applications should note that many implementations only provide line buffering on input from
66286 terminal devices.

66287 **RATIONALE**

66288 None.

66289 **FUTURE DIRECTIONS**

66290 None.

66291 **SEE ALSO**

66292 Section 2.5 (on page 521), *fopen( )*, *setbuf( )*

66293 XBD **<stdio.h>**

66294 **CHANGE HISTORY**

66295 First released in Issue 1. Derived from Issue 1 of the SVID.

66296 **Issue 6**

66297 Extensions beyond the ISO C standard are marked.

66298 The *setvbuf( )* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

66299 **Issue 8**

66300 Austin Group Defect 1144 is applied, changing the [EBADF] error condition.

66301 **NAME**

66302 shm_open — open a shared memory object (**REALTIME**)

66303 **SYNOPSIS**

66304 SHM
```
#include <sys/mman.h>

int shm_open(const char *name, int oflag, mode_t mode);
```

66306 **DESCRIPTION**

66307 The *shm_open*( ) function shall establish a connection between a shared memory object and a file
66308 descriptor. It shall create an open file description that refers to the shared memory object and a
66309 file descriptor that refers to that open file description. The file descriptor shall be allocated as
66310 described in Section 2.6 (on page 525), and can be used by other functions to refer to that shared
66311 memory object. The *name* argument points to a string naming a shared memory object. It is
66312 unspecified whether the name appears in the file system and is visible to other functions that
66313 take pathnames as arguments. The *name* argument conforms to the construction rules for a
66314 pathname, except that the interpretation of <slash> characters other than the leading <slash>
66315 character in *name* is implementation-defined, and that the length limits for the *name* argument
66316 are implementation-defined and need not be the same as the pathname limits {PATH_MAX} and
66317 {NAME_MAX}. If *name* begins with the <slash> character, then processes calling *shm_open*( )
66318 with the same value of *name* refer to the same shared memory object, as long as that name has
66319 not been removed. If *name* does not begin with the <slash> character, the effect is
66320 implementation-defined.

66321 If successful, *shm_open*( ) shall return a file descriptor for the shared memory object. The open
66322 file description is new, and therefore the file descriptor does not share it with any other
66323 processes. It is unspecified whether the file offset is set. The FD_CLOEXEC file descriptor flag
66324 associated with the new file descriptor shall be set.

66325 The file status flags and file access modes of the open file description shall be set according to
66326 the value of *oflag*. The *oflag* argument is the bitwise-inclusive OR of the following flags.
66327 Applications specify exactly one of the first two values (access modes) below in the value of
66328 *oflag*:

66329 O_RDONLY Open for read access only.

66330 O_RDWR Open for read or write access.

66331 Any combination of the remaining flags can be specified in the value of *oflag*:

66332 O_CREAT If the shared memory object exists, this flag has no effect, except as noted
66333 under O_EXCL below. Otherwise, the shared memory object is created. The
66334 user ID of the shared memory object shall be set to the effective user ID of the
66335 process. The group ID of the shared memory object shall be set to the effective
66336 group ID of the process; however, if the *name* argument is visible in the file
66337 system, the group ID may be set to the group ID of the containing directory.
66338 The permission bits of the shared memory object shall be set to the value of
66339 the *mode* argument except those set in the file mode creation mask of the
66340 process. When bits in *mode* other than the file permission bits are set, the effect
66341 is unspecified. The *mode* argument does not affect whether the shared memory
66342 object is opened for reading, for writing, or for both. The shared memory
66343 object has a size of zero.

66344 O_EXCL If O_EXCL and O_CREAT are set, *shm_open*( ) fails if the shared memory
66345 object exists. The check for the existence of the shared memory object and the
66346 creation of the object if it does not exist is atomic with respect to other

processes executing *shm_open*( ) naming the same shared memory object with O_EXCL and O_CREAT set. If O_EXCL is set and O_CREAT is not set, the result is undefined.

O_TRUNC If the shared memory object exists, and it is successfully opened O_RDWR, the object shall be truncated to zero length and the mode and owner shall be unchanged by this function call. The result of using O_TRUNC with O_RDONLY is undefined.

The following functions shall be atomic with respect to each other in the effects specified in POSIX.1-2024 when they operate on shared memory objects:

close( ), *ftruncate*( ), *mmap*( ), *shm_open*( ), *shm_unlink*( )

If two threads each call one of these functions, each call shall either see all of the specified effects of the other call, or none of them. The requirement on the *close*( ) function shall also apply whenever a file descriptor is successfully closed, however caused (for example, as a consequence of calling *close*( ), calling *dup2*( ), or of process termination).

When a shared memory object is created, the state of the shared memory object, including all data associated with the shared memory object, persists until the shared memory object is unlinked and all other references are gone. It is unspecified whether the name and shared memory object state remain valid after a system reboot.

RETURN VALUE

Upon successful completion, the *shm_open*( ) function shall return a non-negative integer representing the file descriptor. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *shm_open*( ) function shall fail if:

[EACCES] The shared memory object exists and the permissions specified by *oflag* are denied, or the shared memory object does not exist and permission to create the shared memory object is denied, or O_TRUNC is specified and write permission is denied.

[EEXIST] O_CREAT and O_EXCL are set and the named shared memory object already exists.

[EINTR] The *shm_open*( ) operation was interrupted by a signal.

[EINVAL] The *shm_open*( ) operation is not supported for the given name.

[EMFILE] All file descriptors available to the process are currently open.

[ENFILE] Too many shared memory objects are currently open in the system.

[ENOENT] O_CREAT is not set and the named shared memory object does not exist.

[ENOSPC] There is insufficient space for the creation of the new shared memory object.

The *shm_open*( ) function may fail if:

[ENAMETOOLONG]
The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI systems, or has a pathname component that is longer than {_POSIX_NAME_MAX} on systems that do not support the XSI option or longer than {_XOPEN_NAME_MAX} on XSI systems.

EXAMPLES

Creating and Mapping a Shared Memory Object

The following code segment demonstrates the use of *shm_open*( ) to create a shared memory object which is then sized using *ftruncate*( ) before being mapped into the process address space using *mmap*( ):

```
#include <unistd.h>
#include <sys/mman.h>
...

#define MAX_LEN 10000
struct region {        /* Defines "structure" of shared memory */
    int len;
    char buf[MAX_LEN];
};
struct region *rptr;
int fd;

/* Create shared memory object and set its size */

fd = shm_open("/myregion", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
if (fd == -1)
    /* Handle error */;

if (ftruncate(fd, sizeof(struct region)) == -1)
    /* Handle error */;

/* Map shared memory object */

rptr = mmap(NULL, sizeof(struct region),
       PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
if (rptr == MAP_FAILED)
    /* Handle error */;

/* Now we can refer to mapped region using fields of rptr;
   for example, rptr->len */
...
```

APPLICATION USAGE

None.

RATIONALE

When the Memory Mapped Files option is supported, the normal *open*( ) call is used to obtain a descriptor to a file to be mapped according to existing practice with *mmap*( ). When the Shared Memory Objects option is supported, the *shm_open*( ) function shall obtain a descriptor to the shared memory object to be mapped.

There is ample precedent for having a file descriptor represent several types of objects. In the POSIX.1-1990 standard, a file descriptor can represent a file, a pipe, a FIFO, a tty, or a directory. Many implementations simply have an operations vector, which is indexed by the file descriptor type and does very different operations. Note that in some cases the file descriptor passed to generic operations on file descriptors is returned by *open*( ) or *creat*( ) and in some cases returned by alternate functions, such as *pipe*( ). The latter technique is used by *shm_open*( ).

Note that such shared memory objects can actually be implemented as mapped files. In both cases, the size can be set after the open using *ftruncate*( ). The *shm_open*( ) function itself does not

create a shared object of a specified size because this would duplicate an extant function that set the size of an object referenced by a file descriptor.

On implementations where memory objects are implemented using the existing file system, the *shm_open*( ) function may be implemented using a macro that invokes *open*( ), and the *shm_unlink*( ) function may be implemented using a macro that invokes *unlink*( ).

For implementations without a permanent file system, the definition of the name of the memory objects is allowed not to survive a system reboot. Note that this allows systems with a permanent file system to implement memory objects as data structures internal to the implementation as well.

On implementations that choose to implement memory objects using memory directly, a *shm_open*( ) followed by an *ftruncate*( ) and *close*( ) can be used to preallocate a shared memory area and to set the size of that preallocation. This may be necessary for systems without virtual memory hardware support in order to ensure that the memory is contiguous.

The set of valid open flags to *shm_open*( ) was restricted to O_RDONLY, O_RDWR, O_CREAT, and O_TRUNC because these could be easily implemented on most memory mapping systems. This volume of POSIX.1-2024 is silent on the results if the implementation cannot supply the requested file access because of implementation-defined reasons, including hardware ones. The O_CLOEXEC open flag is not listed, because all shared memory objects are created with the FD_CLOEXEC flag already set; an application can later use *fcntl*( ) to clear that flag to allow the shared memory file descriptor to be preserved across the *exec* family of functions.

The error conditions [EACCES] and [ENOTSUP] are provided to inform the application that the implementation cannot complete a request.

[EACCES] indicates for implementation-defined reasons, probably hardware-related, that the implementation cannot comply with a requested mode because it conflicts with another requested mode. An example might be that an application desires to open a memory object two times, mapping different areas with different access modes. If the implementation cannot map a single area into a process space in two places, which would be required if different access modes were required for the two areas, then the implementation may inform the application at the time of the second open.

[ENOTSUP] indicates for implementation-defined reasons, probably hardware-related, that the implementation cannot comply with a requested mode at all. An example would be that the hardware of the implementation cannot support write-only shared memory areas.

On all implementations, it may be desirable to restrict the location of the memory objects to specific file systems for performance (such as a RAM disk) or implementation-defined reasons (shared memory supported directly only on certain file systems). The *shm_open*( ) function may be used to enforce these restrictions. There are a number of methods available to the application to determine an appropriate name of the file or the location of an appropriate directory. One way is from the environment via *getenv*( ). Another would be from a configuration file.

This volume of POSIX.1-2024 specifies that memory objects have initial contents of zero when created. This is consistent with current behavior for both files and newly allocated memory. For those implementations that use physical memory, it would be possible that such implementations could simply use available memory and give it to the process uninitialized. This, however, is not consistent with standard behavior for the uninitialized data area, the stack, and of course, files. Finally, it is highly desirable to set the allocated memory to zero for security reasons. Thus, initializing memory objects to zero is required.

FUTURE DIRECTIONS

A future version might require the *shm_open*( ) and *shm_unlink*( ) functions to have semantics similar to normal file system operations.

SEE ALSO

Section 2.6 (on page 525), *close*( ), *dup*( ), *exec*, *fcntl*( ), *mmap*( ), *shmat*( ), *shmctl*( ), *shmdt*( ), *shm_unlink*( ), *umask*( )

XBD **<fcntl.h>**, **<sys/mman.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *shm_open*( ) function is marked as part of the Shared Memory Objects option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Shared Memory Objects option.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/126 is applied, adding the example to the EXAMPLES section.

Issue 7

Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and changing [ENAMETOOLONG] from a ``shall fail'' to a ``may fail'' error.

Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the user ID and group ID of the shared memory object.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0324 [835], XSH/TC2-2008/0325 [835], and XSH/TC2-2008/0326 [835] are applied.

Issue 8

Austin Group Defect 411 is applied, adding a sentence about O_CLOEXEC to the RATIONALE section.

Austin Group Defect 593 is applied, removing a reference to **<fcntl.h>** from the DESCRIPTION section.

Austin Group Defect 695 is applied, adding atomicity requirements to operations on shared memory objects.

NAME

shm_unlink — remove a shared memory object (**REALTIME**)

SYNOPSIS

SHM

```
#include <sys/mman.h>

int shm_unlink(const char *name);
```

DESCRIPTION

The *shm_unlink*( ) function shall remove the name of the shared memory object named by the string pointed to by *name*.

If one or more references to the shared memory object exist when the object is unlinked, the name shall be removed before *shm_unlink*( ) returns, but the removal of the memory object contents shall be postponed until all open and map references to the shared memory object have been removed.

Even if the object continues to exist after the last *shm_unlink*( ), reuse of the name shall subsequently cause *shm_open*( ) to behave as if no shared memory object of this name exists (that is, *shm_open*( ) shall fail if O_CREAT is not set, or shall create a new shared memory object if O_CREAT is set).

RETURN VALUE

Upon successful completion, a value of zero shall be returned. Otherwise, a value of −1 shall be returned and *errno* set to indicate the error. If −1 is returned, the named shared memory object shall not be changed by this function call.

ERRORS

The *shm_unlink*( ) function shall fail if:

[EACCES] Permission is denied to unlink the named shared memory object.

[ENOENT] The named shared memory object does not exist.

The *shm_unlink*( ) function may fail if:

[ENAMETOOLONG]

The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems that do not support the XSI option (XSI) or exceeds {_XOPEN_PATH_MAX} on XSI systems, or has a pathname component that is longer than {_POSIX_NAME_MAX} on systems that do not support the XSI option (XSI) or longer than {_XOPEN_NAME_MAX} on XSI systems. A call to *shm_unlink*( ) with a *name* argument that contains the same shared memory object name as was previously used in a successful *shm_open*( ) call shall not give an [ENAMETOOLONG] error.

EXAMPLES

None.

APPLICATION USAGE

Names of memory objects that were allocated with *open*( ) are deleted with *unlink*( ) in the usual fashion. Names of memory objects that were allocated with *shm_open*( ) are deleted with *shm_unlink*( ). Note that the actual memory object is not destroyed until the last close and unmap on it have occurred if it was already in use.

66549 **RATIONALE**

66550 None.

66551 **FUTURE DIRECTIONS**

66552 A future version might require the *shm_open*( ) and *shm_unlink*( ) functions to have semantics
66553 similar to normal file system operations.

66554 **SEE ALSO**

66555 *close*( ), *mmap*( ), *munmap*( ), *shmat*( ), *shmctl*( ), *shmdt*( ), *shm_open*( )

66556 XBD **<sys/mman.h>**

66557 **CHANGE HISTORY**

66558 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

66559 **Issue 6**

66560 The *shm_unlink*( ) function is marked as part of the Shared Memory Objects option.

66561 In the DESCRIPTION, text is added to clarify that reusing the same name after a *shm_unlink*( )
66562 will not attach to the old shared memory object.

66563 The [ENOSYS] error condition has been removed as stubs need not be provided if an
66564 implementation does not support the Shared Memory Objects option.

66565 **Issue 7**

66566 Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a
66567 ``shall fail'' to a ``may fail'' error.

66568 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

NAME

shmat — XSI shared memory attach operation

SYNOPSIS

XSI
```
#include <sys/shm.h>

void *shmat(int shmid, const void *shmaddr, int shmflg);
```

DESCRIPTION

The *shmat*( ) function operates on XSI shared memory (see XBD Section 3.332, on page 80). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 527).

The *shmat*( ) function attaches the shared memory segment associated with the shared memory identifier specified by *shmid* to the address space of the calling process. The segment is attached at the address specified by one of the following criteria:

- If *shmaddr* is a null pointer, the segment is attached at the first available address as selected by the system.
- If *shmaddr* is not a null pointer and (*shmflg* & SHM_RND) is non-zero, the segment is attached at the address given by ((**char** *)*shmaddr* – ((**uintptr_t**)*shmaddr* % SHMLBA)). The character `'%'` is the C-language remainder operator.
- If *shmaddr* is not a null pointer and (*shmflg* & SHM_RND) is 0, the segment is attached at the address given by *shmaddr*.
- The segment is attached for reading if (*shmflg* & SHM_RDONLY) is non-zero and the calling process has read permission; otherwise, if (*shmflg* & SHM_RDONLY) is 0 and the calling process has read and write permission, the segment is attached for reading and writing.

RETURN VALUE

Upon successful completion, *shmat*( ) shall increment the value of *shm_nattch* in the data structure associated with the shared memory ID of the attached shared memory segment and return the segment's start address. Also, the *shm_atime* timestamp shall be set to the current time, as described in Section 2.7.1 (on page 526).

Otherwise, the shared memory segment shall not be attached, *shmat*( ) shall return SHM_FAILED, and *errno* shall be set to indicate the error.

ERRORS

The *shmat*( ) function shall fail if:

[EACCES] Operation permission is denied to the calling process; see Section 2.7 (on page 526).

[EINVAL] The value of *shmid* is not a valid shared memory identifier, the *shmaddr* is not a null pointer, and the value of ((**char** *)*shmaddr* – ((**uintptr_t**)*shmaddr* % SHMLBA)) is an illegal address for attaching shared memory; or the *shmaddr* is not a null pointer, (*shmflg* & SHM_RND) is 0, and the value of *shmaddr* is an illegal address for attaching shared memory.

[EMFILE] The number of shared memory segments attached to the calling process would exceed the system-imposed limit.

[ENOMEM] The available data space is not large enough to accommodate the shared memory segment.

EXAMPLES

None.

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 526) can be easily modified to use the alternative interfaces.

RATIONALE

The symbol SHM_FAILED is used for the failure return of *shmat*( ) for consistency with MAP_FAILED for *mmap*( ). However, SHM_FAILED is required to have the same value as ((**void** *)(**intptr_t**)–1) to provide backwards compatibility for applications written to earlier versions of this standard, where the failure return was specified as (**void** *)–1. This means that implementations need to ensure that *shmat*( ) cannot return ((**void** *)(**intptr_t**)–1) on a successful call.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7 (on page 526), Section 2.8 (on page 527), *exec*, *exit*( ), *fork*( ), *shmctl*( ), *shmdt*( ), *shmget*( ), *shm_open*( ), *shm_unlink*( )

XBD Section 3.332 (on page 80), **<sys/shm.h>**

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

Moved from SHARED MEMORY to BASE.

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

Issue 6

The Open Group Corrigendum U021/13 is applied.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0572 [345] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0327 [522] is applied.

Issue 8

Austin Group Defect 1237 is applied, changing ``*shmaddr* – . . .'' to ``(**char** *)*shmaddr* – . . .''.

Austin Group Defect 1238 is applied, changing the DESCRIPTION to avoid an ambiguous use of ``it''.

Austin Group Defect 1239 is applied, adding SHM_FAILED.

NAME

shmctl — XSI shared memory control operations

SYNOPSIS

XSI
```
#include <sys/shm.h>

int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

DESCRIPTION

The *shmctl*( ) function operates on XSI shared memory (see XBD Section 3.332, on page 80). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 527).

The *shmctl*( ) function provides a variety of shared memory control operations as specified by *cmd*. The following values for *cmd* are available:

IPC_STAT
: Place the current value of each member of the **shmid_ds** data structure associated with *shmid* into the structure pointed to by *buf*. The contents of the structure are defined in **<sys/shm.h>**.

IPC_SET
: Set the value of the following members of the **shmid_ds** data structure associated with *shmid* to the corresponding value found in the structure pointed to by *buf*:

  `shm_perm.uid`
  `shm_perm.gid`
  `shm_perm.mode` Low-order nine bits.

  Also, the *shm_ctime* timestamp shall be set to the current time, as described in Section 2.7.1 (on page 526).

  IPC_SET can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of *shm_perm.cuid* or *shm_perm.uid* in the **shmid_ds** data structure associated with *shmid*.

IPC_RMID
: Remove the shared memory identifier specified by *shmid* from the system. The shared memory segment and **shmid_ds** data structure associated with it shall be destroyed when all processes with the segment attached have either detached the segment or terminated. If the segment is not attached to any process, it shall be destroyed immediately. IPC_RMID can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of *shm_perm.cuid* or *shm_perm.uid* in the **shmid_ds** data structure associated with *shmid*.

RETURN VALUE

Upon successful completion, *shmctl*( ) shall return 0; otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *shmctl*( ) function shall fail if:

[EACCES]
: The argument *cmd* is equal to IPC_STAT and the calling process does not have read permission; see Section 2.7 (on page 526).

[EINVAL]
: The value of *shmid* is not a valid shared memory identifier, or the value of *cmd* is not a valid command.

66691 [EPERM] The argument *cmd* is equal to IPC_RMID or IPC_SET and the effective user ID
66692 of the calling process is not equal to that of a process with appropriate
66693 privileges and it is not equal to the value of *shm_perm.cuid* or *shm_perm.uid* in
66694 the data structure associated with *shmid*.

66695 The *shmctl*( ) function may fail if:

66696 [EOVERFLOW] The *cmd* argument is IPC_STAT and the *gid* or *uid* value is too large to be
66697 stored in the structure pointed to by the *buf* argument.

66698 **EXAMPLES**

66699 None.

66700 **APPLICATION USAGE**

66701 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
66702 Application developers who need to use IPC should design their applications so that modules
66703 using the IPC routines described in Section 2.7 (on page 526) can be easily modified to use the
66704 alternative interfaces.

66705 **RATIONALE**

66706 None.

66707 **FUTURE DIRECTIONS**

66708 None.

66709 **SEE ALSO**

66710 Section 2.7 (on page 526), Section 2.8 (on page 527), *shmat*( ), *shmdt*( ), *shmget*( ), *shm_open*( ),
66711 *shm_unlink*( )

66712 XBD Section 3.332 (on page 80), **<sys/shm.h>**

66713 **CHANGE HISTORY**

66714 First released in Issue 2. Derived from Issue 2 of the SVID.

66715 **Issue 5**

66716 Moved from SHARED MEMORY to BASE.

66717 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
66718 DIRECTIONS to a new APPLICATION USAGE section.

66719 **Issue 7**

66720 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0573 [345] is applied.

66721 **Issue 8**

66722 Austin Group Defect 1240 is applied, clarifying the description of IPC_RMID.

NAME

shmdt — XSI shared memory detach operation

SYNOPSIS

XSI
```
#include <sys/shm.h>

int shmdt(const void *shmaddr);
```

DESCRIPTION

The *shmdt*( ) function operates on XSI shared memory (see XBD Section 3.332, on page 80). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 527).

The *shmdt*( ) function detaches the shared memory segment located at the address specified by *shmaddr* from the address space of the calling process.

RETURN VALUE

Upon successful completion, *shmdt*( ) shall decrement the value of *shm_nattch* in the data structure associated with the shared memory ID of the attached shared memory segment and return 0. Also, the *shm_dtime* timestamp shall be set to the current time, as described in Section 2.7.1 (on page 526).

Otherwise, the shared memory segment shall not be detached, *shmdt*( ) shall return –1, and *errno* shall be set to indicate the error.

ERRORS

The *shmdt*( ) function shall fail if:

[EINVAL] The value of *shmaddr* is not the data segment start address of a shared memory segment.

EXAMPLES

None.

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 526) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7 (on page 526), Section 2.8 (on page 527), *exec*, *exit*( ), *fork*( ), *shmat*( ), *shmctl*( ), *shmget*( ), *shm_open*( ), *shm_unlink*( )

XBD Section 3.332 (on page 80), **<sys/shm.h>**

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

66762 **Issue 5**

66763 Moved from SHARED MEMORY to BASE.

66764 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
66765 DIRECTIONS to a new APPLICATION USAGE section.

66766 **Issue 7**

66767 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0574 [345] is applied.

NAME

shmget — get an XSI shared memory segment

SYNOPSIS

XSI
```
#include <sys/shm.h>

int shmget(key_t key, size_t size, int shmflg);
```

DESCRIPTION

The *shmget*( ) function operates on XSI shared memory (see XBD Section 3.332, on page 80). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 527).

The *shmget*( ) function shall return the shared memory identifier associated with *key*.

A shared memory identifier, associated data structure, and shared memory segment of at least *size* bytes (see **<sys/shm.h>**) are created for *key* if one of the following is true:

- The argument *key* is equal to IPC_PRIVATE.
- The argument *key* does not already have a shared memory identifier associated with it and (*shmflg* &IPC_CREAT) is non-zero.

Upon creation, the data structure associated with the new shared memory identifier shall be initialized as follows:

- The values of *shm_perm.cuid*, *shm_perm.uid*, *shm_perm.cgid*, and *shm_perm.gid* are set to the effective user ID and effective group ID, respectively, of the calling process.
- The low-order nine bits of *shm_perm.mode* are set to the low-order nine bits of *shmflg*.
- The value of *shm_segsz* is set to the value of *size*.
- The values of *shm_lpid*, *shm_nattch*, *shm_atime*, and *shm_dtime* are set to 0.
- The value of *shm_ctime* is set to the current time, as described in Section 2.7.1 (on page 526).

When the shared memory segment is created, it shall be initialized with all zero values.

RETURN VALUE

Upon successful completion, *shmget*( ) shall return a non-negative integer, namely a shared memory identifier; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *shmget*( ) function shall fail if:

[EACCES] A shared memory identifier exists for *key* but operation permission as specified by the low-order nine bits of *shmflg* would not be granted; see Section 2.7 (on page 526).

[EEXIST] A shared memory identifier exists for the argument *key* but (*shmflg* &IPC_CREAT) &&(*shmflg* &IPC_EXCL) is non-zero.

[EINVAL] A shared memory segment is to be created and the value of size is less than the system-imposed minimum or greater than the system-imposed maximum.

[EINVAL] No shared memory segment is to be created and a shared memory segment exists for *key* but the size of the segment associated with it is less than *size*.

[ENOENT] A shared memory identifier does not exist for the argument *key* and (*shmflg* &IPC_CREAT) is 0.

66808 [ENOMEM] A shared memory identifier and associated shared memory segment are to be
66809 created, but the amount of available physical memory is not sufficient to fill
66810 the request.

66811 [ENOSPC] A shared memory identifier is to be created, but the system-imposed limit on
66812 the maximum number of allowed shared memory identifiers system-wide
66813 would be exceeded.

66814 **EXAMPLES**

66815 None.

66816 **APPLICATION USAGE**

66817 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
66818 Application developers who need to use IPC should design their applications so that modules
66819 using the IPC routines described in Section 2.7 (on page 526) can be easily modified to use the
66820 alternative interfaces.

66821 **RATIONALE**

66822 None.

66823 **FUTURE DIRECTIONS**

66824 None.

66825 **SEE ALSO**

66826 Section 2.7 (on page 526), Section 2.8 (on page 527), *ftok*( ), *shmat*( ), *shmctl*( ), *shmdt*( ), *shm_open*( ),
66827 *shm_unlink*( )

66828 XBD Section 3.332 (on page 80), **<sys/shm.h>**

66829 **CHANGE HISTORY**

66830 First released in Issue 2. Derived from Issue 2 of the SVID.

66831 **Issue 5**

66832 Moved from SHARED MEMORY to BASE.

66833 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
66834 DIRECTIONS to a new APPLICATION USAGE section.

66835 **Issue 7**

66836 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0575 [345], XSH/TC1-2008/0576 [363],
66837 and XSH/TC1-2008/0577 [344] are applied.

66838 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0328 [640] is applied.

66839 **NAME**

66840 shutdown — shut down socket send and receive operations

66841 **SYNOPSIS**

66842 `#include <sys/socket.h>`

66843 `int shutdown(int socket, int how);`

66844 **DESCRIPTION**

66845 The *shutdown*( ) function shall cause all or part of a full-duplex connection on the socket
66846 associated with the file descriptor *socket* to be shut down.

66847 The *shutdown*( ) function takes the following arguments:

66848 *socket* Specifies the file descriptor of the socket.

66849 *how* Specifies the type of shutdown. The values are as follows:

66850 SHUT_RD Disables further receive operations.

66851 SHUT_WR Disables further send operations.

66852 SHUT_RDWR Disables further send and receive operations.

66853 The *shutdown*( ) function disables subsequent send and/or receive operations on a socket,
66854 depending on the value of the *how* argument.

66855 **RETURN VALUE**

66856 Upon successful completion, *shutdown*( ) shall return 0; otherwise, –1 shall be returned and *errno*
66857 set to indicate the error.

66858 **ERRORS**

66859 The *shutdown*( ) function shall fail if:

66860 [EBADF] The *socket* argument is not a valid file descriptor.

66861 [EINVAL] The *how* argument is invalid.

66862 [ENOTCONN] The socket is not connected.

66863 [ENOTSOCK] The *socket* argument does not refer to a socket.

66864 The *shutdown*( ) function may fail if:

66865 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

66866 **EXAMPLES**

66867 None.

66868 **APPLICATION USAGE**

66869 The file descriptor remains open after *shutdown*( ) returns to the calling application.

66870 **RATIONALE**

66871 None.

66872 **FUTURE DIRECTIONS**

66873 None.

66874 **SEE ALSO**

66875 *close*( ), *getsockopt*( ), *pselect*( ), *read*( ), *recv*( ), *recvfrom*( ), *recvmsg*( ), *send*( ), *sendto*( ), *setsockopt*( ),
66876 *socket*( ), *write*( )

66877 XBD **<sys/socket.h>**

66878 **CHANGE HISTORY**

66879 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

66880 **Issue 8**

66881 Austin Group Defect 1475 is applied, changing the APPLICATION USAGE section and adding
66882 *close*( ) to the SEE ALSO section.

NAME

sig2str, str2sig — translate between signal names and numbers

SYNOPSIS

CX
```
#include <signal.h>

int sig2str(int signum, char *str);
int str2sig(const char *restrict str, int *restrict pnum);
```

DESCRIPTION

The *sig2str*( ) function shall translate the signal number specified by *signum* to a signal name and shall store this string in the location specified by *str*. The application shall ensure that *str* points to a location that can store the string including the terminating null byte. The symbolic constant SIG2STR_MAX defined in **<signal.h>** gives the maximum number of bytes required.

If *signum* is equal to 0, the behavior is unspecified.

If *signum* is equal to one of the symbolic constants listed in the table of signal numbers in **<signal.h>**, the stored signal name shall be the name of the symbolic constant without the **SIG** prefix.

If *signum* is equal to SIGRTMIN or SIGRTMAX, the stored string shall be "RTMIN" or "RTMAX", respectively.

If *signum* is between SIGRTMIN+1 and (SIGRTMIN+SIGRTMAX)/2 inclusive, the stored string shall be of the form "RTMIN+*n*", where *n* is the shortest decimal representation of the value of *signum*–SIGRTMIN.

If *signum* is between (SIGRTMIN+SIGRTMAX)/2 + 1 and SIGRTMAX–1 inclusive, the stored string shall be either of the form "RTMIN+*n*" or of the form "RTMAX–*m*", where *n* is the shortest decimal representation of the value of *signum*–SIGRTMIN and *m* is the shortest decimal representation of the value of SIGRTMAX–*signum*.

If *signum* is a valid, supported signal number, is either less than SIGRTMIN or greater than SIGRTMAX, and is not equal to one of the symbolic constants listed in the table of signal numbers in **<signal.h>**, the stored string shall uniquely identify the signal number *signum* in an unspecified manner.

The *str2sig*( ) function shall translate the signal name in the string pointed to by *str* to a signal number and shall store this value in the location specified by *pnum*.

If *str* points to a string containing the name of one of the symbolic constants listed in the table of signal numbers in **<signal.h>**, without the **SIG** prefix, the stored signal number shall be equal to the value of the symbolic constant.

If *str* points to the string "RTMIN" or "RTMAX", the stored value shall be equal to SIGRTMIN or SIGRTMAX, respectively.

If *str* points to a string of the form "RTMIN+*n*", where *n* is a decimal representation of a number between 1 and SIGRTMAX–SIGRTMIN–1 inclusive, the stored value shall be equal to SIGRTMIN+n.

If *str* points to a string of the form "RTMAX–*n*", where *n* is a decimal representation of a number between 1 and SIGRTMAX–SIGRTMIN–1 inclusive, the stored value shall be equal to SIGRTMAX–*n*.

If *str* points to a string containing a decimal representation of a valid, supported signal number, the value stored in the location pointed to by *pnum* shall be equal to that number.

If *str* points to a string containing a decimal representation of the value 0 and the string was not returned by a previous successful call to *sig2str*( ) with a *signum* argument of 0, the behavior is unspecified.

If *str* points to a string returned by a previous successful call to *sig2str*(*signum*,*str*), the value stored in the location pointed to by *pnum* shall be equal to *signum*.

If *str* points to a string that does not meet any of the above criteria, *str2sig*( ) shall store a value in the location pointed to by *pnum* if and only if it recognizes the string as an additional implementation-dependent form of signal name.

RETURN VALUE

If *signum* is a valid, supported signal number (that is, one for which *kill*( ) does not return –1 with *errno* set to [EINVAL]), the *sig2str*( ) function shall return 0; otherwise, if *signum* is not equal to 0, it shall return –1.

If *str2sig*( ) stores a value in the location pointed to by *pnum*, it shall return 0; otherwise, it shall return –1.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Historical versions of these functions translated a *signum* value 0 to `"EXIT"` (and vice versa), so that they could be used by the shell for the *trap* utility. When adding the functions to this standard, the standard developers felt that they should be aimed at more general-purpose use, and consequently requiring this behavior did not seem appropriate and so the behavior in this case has been made unspecified.

FUTURE DIRECTIONS

None.

SEE ALSO

kill( ), *sigaction*( ), *strsignal*( )

XBD **<signal.h>**

CHANGE HISTORY

First released in Issue 8.

66959 **NAME**

66960 sigaction — examine and change a signal action

66961 **SYNOPSIS**

66962 CX
```
#include <signal.h>

int sigaction(int sig, const struct sigaction *restrict act,
    struct sigaction *restrict oact);
```

66965 **DESCRIPTION**

66966 The *sigaction*( ) function allows the calling process to examine and/or specify the action to be
66967 associated with a specific signal. The argument *sig* specifies the signal; acceptable values are
66968 defined in **<signal.h>**.

66969 The structure **sigaction**, used to describe an action to be taken, is defined in the **<signal.h>**
66970 header to include at least the following members:

66971

Member Type	Member Name	Description
void(*) (int)	*sa_handler*	Pointer to a signal-catching function or one of the macros SIG_IGN or SIG_DFL.
sigset_t	*sa_mask*	Additional set of signals to be blocked during execution of signal-catching function.
int	*sa_flags*	Special flags to affect behavior of signal.
void(*) (int, siginfo_t *, void *)	*sa_sigaction*	Pointer to a signal-catching function.

66980 The storage occupied by *sa_handler* and *sa_sigaction* may overlap, and a conforming application
66981 shall not use both simultaneously.

66982 If the argument *act* is not a null pointer, it points to a structure specifying the action to be
66983 associated with the specified signal. If the argument *oact* is not a null pointer, the action
66984 previously associated with the signal is stored in the location pointed to by the argument *oact*. If
66985 the argument *act* is a null pointer, signal handling is unchanged; thus, the call can be used to
66986 enquire about the current handling of a given signal. The SIGKILL and SIGSTOP signals shall
66987 not be added to the signal mask using this mechanism; this restriction shall be enforced by the
66988 system without causing an error to be indicated.

66989 If the SA_SIGINFO flag (see below) is cleared in the *sa_flags* field of the **sigaction** structure, the
66990 *sa_handler* field identifies the action to be associated with the specified signal. If the
66991 SA_SIGINFO flag is set in the *sa_flags* field, the *sa_sigaction* field specifies a signal-catching
66992 function.

66993 The *sa_flags* field can be used to modify the behavior of the specified signal.

66994 The following flags, defined in the **<signal.h>** header, can be set in *sa_flags*:

66995 XSI SA_NOCLDSTOP Do not generate SIGCHLD when children stop or stopped children
66996 continue.

66997 If *sig* is SIGCHLD and the SA_NOCLDSTOP flag is not set in *sa_flags*, and
66998 the implementation supports the SIGCHLD signal, then a SIGCHLD
66999 signal shall be generated for the calling process whenever any of its child
67000 XSI processes stop and a SIGCHLD signal may be generated for the calling
67001 process whenever any of its stopped child processes are continued. If *sig*
67002 is SIGCHLD and the SA_NOCLDSTOP flag is set in *sa_flags*, then the
67003 implementation shall not generate a SIGCHLD signal in this way.

67004 XSI SA_ONSTACK If set and an alternate signal stack has been declared with *sigaltstack*( ), the
67005 signal shall be delivered to the calling process on that stack. Otherwise,
67006 the signal shall be delivered on the current stack.

67007 SA_RESETHAND If set, the disposition of the signal shall be reset to SIG_DFL and the
67008 SA_SIGINFO flag shall be cleared on entry to the signal handler.

67009 **Note:** SIGILL and SIGTRAP cannot be automatically reset when delivered;
67010 the system silently enforces this restriction.

67011 Otherwise, the disposition of the signal shall not be modified on entry to
67012 the signal handler.

67013 In addition, if this flag is set, *sigaction*( ) may behave as if the
67014 SA_NODEFER flag were also set.

67015 SA_RESTART This flag affects the behavior of interruptible functions; that is, those
67016 specified to fail with *errno* set to [EINTR]. If set, and a function specified
67017 as interruptible is interrupted by this signal, the function shall restart and
67018 shall not fail with [EINTR] unless otherwise specified. If an interruptible
67019 function which uses a timeout is restarted, the duration of the timeout
67020 following the restart is set to an unspecified value that does not exceed
67021 the original timeout value. If the flag is not set, interruptible functions
67022 interrupted by this signal shall fail with *errno* set to [EINTR].

67023 SA_SIGINFO If cleared and the signal is caught, the signal-catching function shall be
67024 entered as:

67025 `void func(int signo);`

67026 where *signo* is the only argument to the signal-catching function. In this
67027 case, the application shall use the *sa_handler* member to describe the
67028 signal-catching function and the application shall not modify the
67029 *sa_sigaction* member.

67030 If SA_SIGINFO is set and the signal is caught, the signal-catching
67031 function shall be entered as:

67032 `void func(int signo, siginfo_t *info, void *context);`

67033 where two additional arguments are passed to the signal-catching
67034 function. The second argument shall point to an object of type **siginfo_t**
67035 explaining the reason why the signal was generated; the third argument
67036 can be cast to a pointer to an object of type **ucontext_t** to refer to the
67037 receiving thread's context that was interrupted when the signal was
67038 delivered. In this case, the application shall use the *sa_sigaction* member to
67039 describe the signal-catching function and the application shall not modify
67040 the *sa_handler* member.

67041 The *si_signo* member contains the system-generated signal number.

67042 XSI The *si_errno* member may contain implementation-defined additional
67043 error information; if non-zero, it contains an error number identifying the
67044 condition that caused the signal to be generated.

67045 The *si_code* member contains a code identifying the cause of the signal, as
67046 described in Section 2.4.3 (on page 516).

67047 XSI SA_NOCLDWAIT If *sig* does not equal SIGCHLD, the behavior is unspecified. Otherwise,
67048 the behavior of the SA_NOCLDWAIT flag is as specified in Consequences
67049 of Process Termination (on page 568).

67050 SA_NODEFER If set and *sig* is caught, *sig* shall not be added to the thread's signal mask
67051 on entry to the signal handler unless it is included in *sa_mask*. Otherwise,
67052 *sig* shall always be added to the thread's signal mask on entry to the
67053 signal handler.

67054 When a signal is caught by a signal-catching function installed by *sigaction*( ), a new signal mask
67055 is calculated and installed for the duration of the signal-catching function (or until a call to either
67056 *sigprocmask*( ) or *sigsuspend*( ) is made). This mask is formed by taking the union of the current
67057 signal mask and the value of the *sa_mask* for the signal being delivered, and unless
67058 SA_NODEFER or SA_RESETHAND is set, then including the signal being delivered. If and
67059 when the user's signal handler returns normally, the original signal mask is restored.

67060 Once an action is installed for a specific signal, it shall remain installed until another action is
67061 explicitly requested (by another call to *sigaction*( )), until the SA_RESETHAND flag causes
67062 resetting of the handler, or until one of the *exec* functions is called.

67063 If the previous action for *sig* had been established by *signal*( ), the values of the fields returned in
67064 the structure pointed to by *oact* are unspecified, and in particular *oact*->*sa_handler* is not
67065 necessarily the same value passed to *signal*( ). However, if a pointer to the same structure or a
67066 copy thereof is passed to a subsequent call to *sigaction*( ) via the *act* argument, handling of the
67067 signal shall be as if the original call to *signal*( ) were repeated.

67068 If *sigaction*( ) fails, no new signal handler is installed.

67069 It is unspecified whether an attempt to set the action for a signal that cannot be caught or
67070 ignored to SIG_DFL is ignored or causes an error to be returned with *errno* set to [EINVAL].

67071 If SA_SIGINFO is not set in *sa_flags*, then the disposition of subsequent occurrences of *sig* when
67072 it is already pending is implementation-defined; the signal-catching function shall be invoked
67073 with a single argument. If SA_SIGINFO is set in *sa_flags*, then subsequent occurrences of *sig*
67074 generated by *sigqueue*( ) or as a result of any signal-generating function that supports the
67075 specification of an application-defined value (when *sig* is already pending) shall be queued in
67076 FIFO order until delivered or accepted; the signal-catching function shall be invoked with three
67077 arguments. The application specified value is passed to the signal-catching function as the
67078 *si_value* member of the **siginfo_t** structure.

67079 The result of the use of *sigaction*( ) and a *sigwait*( ) function concurrently within a process on the
67080 same signal is unspecified.

67081 **RETURN VALUE**

67082 Upon successful completion, *sigaction*( ) shall return 0; otherwise, –1 shall be returned, *errno* shall
67083 be set to indicate the error, and no new signal-catching function shall be installed.

ERRORS

The *sigaction*( ) function shall fail if:

[EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a signal that cannot be caught or ignore a signal that cannot be ignored.

The *sigaction*( ) function may fail if:

[EINVAL] An attempt was made to set the action to SIG_DFL for a signal that cannot be caught or ignored (or both).

In addition, on systems that do not support the XSI option, the *sigaction*( ) function may fail if the SA_SIGINFO flag is set in the *sa_flags* field of the **sigaction** structure for a signal not in the range SIGRTMIN to SIGRTMAX.

EXAMPLES

Establishing a Signal Handler

The following example demonstrates the use of *sigaction*( ) to establish a handler for the SIGINT signal.

```
#include <signal.h>

static void handler(int signum)
{
    /* Take appropriate actions for signal delivery */
}

int main(void)
{
    struct sigaction sa;

    sa.sa_handler = handler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = SA_RESTART; /* Restart functions if
                                 interrupted by handler */
    if (sigaction(SIGINT, &sa, NULL) == -1)
        /* Handle error */;

    /* Further code */
}
```

APPLICATION USAGE

The *sigaction*( ) function supersedes the *signal*( ) function, and should be used in preference. In particular, *sigaction*( ) and *signal*( ) should not be used in the same process to control the same signal. The behavior of async-signal-safe functions, as defined in their respective DESCRIPTION sections, is as specified by this volume of POSIX.1-2024, regardless of invocation from a signal-catching function. This is the only intended meaning of the statement that async-signal-safe functions may be used in signal-catching functions without restrictions. Applications must still consider all effects of such functions on such things as data structures, files, and process state. In particular, application developers need to consider the restrictions on interactions when interrupting *sleep*( ) and interactions among multiple handles for a file description. The fact that any specific function is listed as async-signal-safe does not necessarily mean that invocation of that function from a signal-catching function is recommended.

In order to prevent errors arising from interrupting non-async-signal-safe function calls, applications should protect calls to these functions either by blocking the appropriate signals or

through the use of some programmatic semaphore (see *semget( )*, *sem_init( )*, *sem_open( )*, and so on). Note in particular that even the ``safe'' functions may modify *errno*; the signal-catching function, if not executing as an independent thread, should save and restore its value in order to avoid the possibility that delivery of a signal in between an error return from a function that sets *errno* and the subsequent examination of *errno* could result in the signal-catching function changing the value of *errno*. Naturally, the same principles apply to the async-signal-safety of application routines and asynchronous data access. Note that *longjmp( )* and *siglongjmp( )* are not in the list of async-signal-safe functions. This is because the code executing after *longjmp( )* and *siglongjmp( )* can call any unsafe functions with the same danger as calling those unsafe functions directly from the signal handler. Applications that use *longjmp( )* and *siglongjmp( )* from within signal handlers require rigorous protection in order to be portable. Many of the other functions that are excluded from the list are traditionally implemented using either *malloc( )* or *free( )* functions or the standard I/O library, both of which traditionally use data structures in a non-async-signal-safe manner. Since any combination of different functions using a common data structure can cause async-signal-safety problems, this volume of POSIX.1-2024 does not define the behavior when any unsafe function is called in a signal handler that interrupts an unsafe function.

Usually, the signal is executed on the stack that was in effect before the signal was delivered. An alternate stack may be specified to receive a subset of the signals being caught.

When the signal handler returns, the receiving thread resumes execution at the point it was interrupted unless the signal handler makes other arrangements. If *longjmp( )* is used to leave the signal handler, then the signal mask must be explicitly restored.

This volume of POSIX.1-2024 defines the third argument of a signal handling function when SA_SIGINFO is set as a **void** * instead of a **ucontext_t** *, but without requiring type checking. New applications should explicitly cast the third argument of the signal handling function to **ucontext_t** *.

The BSD optional four argument signal handling function is not supported by this volume of POSIX.1-2024. The BSD declaration would be:

```
void handler(int sig, int code, struct sigcontext *scp,
    char *addr);
```

where *sig* is the signal number, *code* is additional information on certain signals, *scp* is a pointer to the **sigcontext** structure, and *addr* is additional address information. Much the same information is available in the objects pointed to by the second argument of the signal handler specified when SA_SIGINFO is set.

Since the *sigaction( )* function is allowed but not required to set SA_NODEFER when the application sets the SA_RESETHAND flag, applications which depend on the SA_RESETHAND functionality for the newly installed signal handler must always explicitly set SA_NODEFER when they set SA_RESETHAND in order to be portable.

See also the rationale for Realtime Signal Generation and Delivery in XRAT Section B.2.4.2 (on page 3749).

RATIONALE

Although this volume of POSIX.1-2024 requires that signals that cannot be ignored shall not be added to the signal mask when a signal-catching function is entered, there is no explicit requirement that subsequent calls to *sigaction( )* reflect this in the information returned in the *oact* argument. In other words, if SIGKILL is included in the *sa_mask* field of *act*, it is unspecified whether or not a subsequent call to *sigaction( )* returns with SIGKILL included in the *sa_mask* field of *oact*.

The SA_NOCLDSTOP flag, when supplied in the *act->sa_flags* parameter, allows overloading SIGCHLD with the System V semantics that each SIGCLD signal indicates a single terminated child. Most conforming applications that catch SIGCHLD are expected to install signal-catching functions that repeatedly call the *waitpid*( ) function with the WNOHANG flag set, acting on each child for which status is returned, until *waitpid*( ) returns zero. If stopped children are not of interest, the use of the SA_NOCLDSTOP flag can prevent the overhead from invoking the signal-catching routine when they stop.

Some historical implementations also define other mechanisms for stopping processes, such as the *ptrace*( ) function. These implementations usually do not generate a SIGCHLD signal when processes stop due to this mechanism; however, that is beyond the scope of this volume of POSIX.1-2024.

This volume of POSIX.1-2024 requires that calls to *sigaction*( ) that supply a NULL *act* argument succeed, even in the case of signals that cannot be caught or ignored (that is, SIGKILL or SIGSTOP). The System V *signal*( ) and BSD *sigvec*( ) functions return [EINVAL] in these cases and, in this respect, their behavior varies from *sigaction*( ).

This volume of POSIX.1-2024 requires that *sigaction*( ) properly save and restore a signal action set up by the ISO C standard *signal*( ) function. However, there is no guarantee that the reverse is true, nor could there be given the greater amount of information conveyed by the **sigaction** structure. Because of this, applications should avoid using both functions for the same signal in the same process. Since this cannot always be avoided in case of general-purpose library routines, they should always be implemented with *sigaction*( ).

It was intended that the *signal*( ) function should be implementable as a library routine using *sigaction*( ).

The POSIX Realtime Extension extends the *sigaction*( ) function as specified by the POSIX.1-1990 standard to allow the application to request on a per-signal basis via an additional signal action flag that the extra parameters, including the application-defined signal value, if any, be passed to the signal-catching function.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 513), *exec*, *_Exit*( ), *kill*( ), *longjmp*( ), *pthread_sigmask*( ), *raise*( ), *semget*( ), *sem_init*( ), *sem_open*( ), *sig2str*( ), *sigaddset*( ), *sigaltstack*( ), *sigdelset*( ), *sigemptyset*( ), *sigfillset*( ), *sigismember*( ), *signal*( ), *sigsuspend*( ), *wait*( ), *waitid*( )

XBD **<signal.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and POSIX Threads Extension.

In the DESCRIPTION, the second argument to *func* when SA_SIGINFO is set is no longer permitted to be NULL, and the description of permitted **siginfo_t** contents is expanded by reference to **<signal.h>**.

Since the X/OPEN UNIX Extension functionality is now folded into the BASE, the [ENOTSUP] error is deleted.

Issue 6

The Open Group Corrigendum U028/7 is applied. In the paragraph entitled ``Signal Effects on Other Functions'', a reference to *sigpending*( ) is added.

In the DESCRIPTION, the text ``Signal Generation and Delivery'', ``Signal Actions'', and ``Signal Effects on Other Functions'' are moved to a separate section of this volume of POSIX.1-2024.

Text describing functionality from the Realtime Signals Extension option is marked.

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- The [ENOTSUP] error condition is added.

The normative text is updated to avoid use of the term ``must'' for application requirements.

The **restrict** keyword is added to the *sigaction*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

References to the *wait3*( ) function are removed.

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/57 is applied, changing text in the table describing the **sigaction** structure.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/127 is applied, removing text from the DESCRIPTION duplicated later in the same section.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/128 is applied, updating the DESCRIPTION and APPLICATION USAGE sections. Changes are made to refer to the thread rather than the process.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/129 is applied, adding the example to the EXAMPLES section.

Issue 7

Austin Group Interpretation 1003.1-2001 #004 is applied.

Austin Group Interpretations 1003.1-2001 #065 and #084 are applied, clarifying the role of the SA_NODEFER flag with respect to the signal mask, and clarifying the SA_RESTART flag for interrupted functions which use timeouts.

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XSH-ERN-167 is applied, updating the APPLICATION USAGE section.

SD5-XSH-ERN-172 is applied, updating the DESCRIPTION to make optional the requirement that when the SA_RESETHAND flag is set, *sigaction*( ) shall behave as if the SA_NODEFER flag were also set.

Functionality relating to the Realtime Signals Extension option is moved to the Base.

The description of the *si_code* member is replaced with a reference to Section 2.4.3 (on page 516).

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0578 [66] and XSH/TC1-2008/0579 [140] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0329 [690] and XSH/TC2-2008/0330 [491] are applied.

67258 **Issue 8**

67259 Austin Group Defect 1138 is applied, adding *sig2str*( ) to the SEE ALSO section.

67260 Austin Group Defect 1195 is applied, changing ``main()'' to ``main(void)''.

67261 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

NAME

sigaddset — add a signal to a signal set

SYNOPSIS

CX
```
#include <signal.h>

int sigaddset(sigset_t *set, int signo);
```

DESCRIPTION

The *sigaddset*( ) function adds the individual signal specified by the *signo* to the signal set pointed to by *set*.

Applications shall call either *sigemptyset*( ) or *sigfillset*( ) at least once for each object of type **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is nonetheless supplied as an argument to any of *pthread_sigmask*( ), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigismember*( ), *sigpending*( ), *sigprocmask*( ), *sigsuspend*( ), *sigtimedwait*( ), *sigwait*( ), or *sigwaitinfo*( ), the results are undefined.

RETURN VALUE

Upon successful completion, *sigaddset*( ) shall return 0; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *sigaddset*( ) function may fail if:

[EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 513), *pthread_sigmask*( ), *sigaction*( ), *sigdelset*( ), *sigemptyset*( ), *sigfillset*( ), *sigismember*( ), *sigpending*( ), *sigsuspend*( )

XBD **<signal.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in previous issues.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

67302 **NAME**

67303 sigaltstack — set and get signal alternate stack context

67304 **SYNOPSIS**

67305 XSI
```
#include <signal.h>

int sigaltstack(const stack_t *restrict ss, stack_t *restrict oss);
```

67306

67307 **DESCRIPTION**

67308 The *sigaltstack*( ) function allows a process to define and examine the state of an alternate stack
67309 for signal handlers for the current thread. Signals that have been explicitly declared to execute
67310 on the alternate stack shall be delivered on the alternate stack.

67311 If *ss* is not a null pointer, it points to a **stack_t** structure that specifies the alternate signal stack
67312 that shall take effect upon return from *sigaltstack*( ). The *ss_flags* member specifies the new stack
67313 state. If it is set to SS_DISABLE, the stack is disabled and *ss_sp* and *ss_size* are ignored.
67314 Otherwise, the stack shall be enabled, and the *ss_sp* and *ss_size* members specify the new address
67315 and size of the stack.

67316 The range of addresses starting at *ss_sp* up to but not including *ss_sp*+*ss_size* is available to the
67317 implementation for use as the stack. This function makes no assumptions regarding which end
67318 is the stack base and in which direction the stack grows as items are pushed.

67319 If *oss* is not a null pointer, upon successful completion it shall point to a **stack_t** structure that
67320 specifies the alternate signal stack that was in effect prior to the call to *sigaltstack*( ). The *ss_sp*
67321 and *ss_size* members specify the address and size of that stack. The *ss_flags* member specifies the
67322 stack's state, and may contain one of the following values:

67323 SS_ONSTACK The process is currently executing on the alternate signal stack. Attempts to
67324 modify the alternate signal stack while the process is executing on it fail. This
67325 flag shall not be modified by processes.

67326 SS_DISABLE The alternate signal stack is currently disabled.

67327 The value SIGSTKSZ is a system default specifying the number of bytes that would be used to
67328 cover the usual case when manually allocating an alternate stack area. The value MINSIGSTKSZ
67329 is defined to be the minimum stack size for a signal handler. In computing an alternate stack
67330 size, a program should add that amount to its stack requirements to allow for the system
67331 implementation overhead. The constants SS_ONSTACK, SS_DISABLE, SIGSTKSZ, and
67332 MINSIGSTKSZ are defined in **<signal.h>**.

67333 After a successful call to one of the *exec* functions, there are no alternate signal stacks in the new
67334 process image.

67335 In some implementations, a signal (whether or not indicated to execute on the alternate stack)
67336 shall always execute on the alternate stack if it is delivered while another signal is being caught
67337 using the alternate stack.

67338 Use of this function by library threads that are not bound to kernel-scheduled entities results in
67339 undefined behavior.

67340 **RETURN VALUE**

67341 Upon successful completion, *sigaltstack*( ) shall return 0; otherwise, it shall return –1 and set *errno*
67342 to indicate the error.

67343 **ERRORS**

67344 The *sigaltstack*( ) function shall fail if:

67345 [EINVAL] The *ss* argument is not a null pointer, and the *ss_flags* member pointed to by *ss*
67346 has SS_ONSTACK or invalid flags set.

67347 [ENOMEM] The size of the alternate stack area is less than MINSIGSTKSZ.

67348 [EPERM] An attempt was made to modify an active stack.

67349 **EXAMPLES**

67350 **Allocating Memory for an Alternate Stack**

67351 The following example illustrates a method for allocating memory for an alternate stack.

```
67352 #include <signal.h>
67353 ...
67354 if ((sigstk.ss_sp = malloc(SIGSTKSZ)) == NULL)
67355     /* Error return. */
67356 sigstk.ss_size = SIGSTKSZ;
67357 sigstk.ss_flags = 0;
67358 if (sigaltstack(&sigstk,(stack_t *)0) < 0)
67359     perror("sigaltstack");
```

67360 **APPLICATION USAGE**

67361 On some implementations, stack space is automatically extended as needed. On those
67362 implementations, automatic extension is typically not available for an alternate stack. If the stack
67363 overflows, the behavior is undefined.

67364 **RATIONALE**

67365 None.

67366 **FUTURE DIRECTIONS**

67367 None.

67368 **SEE ALSO**

67369 Section 2.4 (on page 513), *exec*, *sigaction*( ), *sigsetjmp*( )

67370 XBD **<signal.h>**

67371 **CHANGE HISTORY**

67372 First released in Issue 4, Version 2.

67373 **Issue 5**

67374 Moved from X/OPEN UNIX extension to BASE.

67375 The last sentence of the DESCRIPTION was included as an APPLICATION USAGE note in
67376 previous issues.

67377 **Issue 6**

67378 The normative text is updated to avoid use of the term ``must'' for application requirements.

67379 The **restrict** keyword is added to the *sigaltstack*( ) prototype for alignment with the
67380 ISO/IEC 9899: 1999 standard.

67381 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/58 is applied, updating the first sentence
67382 to include ``for the current thread''.

67383 **Issue 8**
67384 Austin Group Defect 1187 is applied, changing the description of the [EINVAL] error.

NAME

sigdelset — delete a signal from a signal set

SYNOPSIS

CX
```
#include <signal.h>

int sigdelset(sigset_t *set, int signo);
```

DESCRIPTION

The *sigdelset*( ) function deletes the individual signal specified by *signo* from the signal set pointed to by *set*.

Applications should call either *sigemptyset*( ) or *sigfillset*( ) at least once for each object of type **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is nonetheless supplied as an argument to any of *pthread_sigmask*( ), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigismember*( ), *sigpending*( ), *sigprocmask*( ), *sigsuspend*( ), *sigtimedwait*( ), *sigwait*( ), or *sigwaitinfo*( ), the results are undefined.

RETURN VALUE

Upon successful completion, *sigdelset*( ) shall return 0; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *sigdelset*( ) function may fail if:

[EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal number.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 513), *pthread_sigmask*( ), *sigaction*( ), *sigaddset*( ), *sigemptyset*( ), *sigfillset*( ), *sigismember*( ), *sigpending*( ), *sigsuspend*( )

XBD **<signal.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in previous issues.

Issue 6

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

NAME

sigemptyset — initialize and empty a signal set

SYNOPSIS

CX
```
#include <signal.h>

int sigemptyset(sigset_t *set);
```

DESCRIPTION

The *sigemptyset*( ) function initializes the signal set pointed to by *set*, such that all signals defined in POSIX.1-2024 are excluded.

RETURN VALUE

Upon successful completion, *sigemptyset*( ) shall return 0; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The implementation of the *sigemptyset*( ) (or *sigfillset*( )) function could quite trivially clear (or set) all the bits in the signal set. Alternatively, it would be reasonable to initialize part of the structure, such as a version field, to permit binary-compatibility between releases where the size of the set varies. For such reasons, either *sigemptyset*( ) or *sigfillset*( ) must be called prior to any other use of the signal set, even if such use is read-only (for example, as an argument to *sigpending*( )). This function is not intended for dynamic allocation.

The *sigfillset*( ) and *sigemptyset*( ) functions require that the resulting signal set include (or exclude) all the signals defined in this volume of POSIX.1-2024. Although it is outside the scope of this volume of POSIX.1-2024 to place this requirement on signals that are implemented as extensions, it is recommended that implementation-defined signals also be affected by these functions. However, there may be a good reason for a particular signal not to be affected. For example, blocking or ignoring an implementation-defined signal may have undesirable side-effects, whereas the default action for that signal is harmless. In such a case, it would be preferable for such a signal to be excluded from the signal set returned by *sigfillset*( ).

In early proposals there was no distinction between invalid and unsupported signals (the names of optional signals that were not supported by an implementation were not defined by that implementation). The [EINVAL] error was thus specified as a required error for invalid signals. With that distinction, it is not necessary to require implementations of these functions to determine whether an optional signal is actually supported, as that could have a significant performance impact for little value. The error could have been required for invalid signals and optional for unsupported signals, but this seemed unnecessarily complex. Thus, the error is optional in both cases.

FUTURE DIRECTIONS

None.

67467 **SEE ALSO**

67468 Section 2.4 (on page 513), *pthread_sigmask( )*, *sigaction( )*, *sigaddset( )*, *sigdelset( )*, *sigfillset( )*,
67469 *sigismember( )*, *sigpending( )*, *sigsuspend( )*

67470 XBD **<signal.h>**

67471 **CHANGE HISTORY**

67472 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

67473 **Issue 6**

67474 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an
67475 extension over the ISO C standard.

NAME

sigfillset — initialize and fill a signal set

SYNOPSIS

CX
```
#include <signal.h>

int sigfillset(sigset_t *set);
```

DESCRIPTION

The *sigfillset*( ) function shall initialize the signal set pointed to by *set*, such that all signals defined in this volume of POSIX.1-2024 are included.

RETURN VALUE

Upon successful completion, *sigfillset*( ) shall return 0; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to *sigemptyset*( ) (on page 2055).

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 513), *pthread_sigmask*( ), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigemptyset*( ), *sigismember*( ), *sigpending*( ), *sigsuspend*( )

XBD **<signal.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

NAME

sigismember — test for a signal in a signal set

SYNOPSIS

CX
```
#include <signal.h>

int sigismember(const sigset_t *set, int signo);
```

DESCRIPTION

The *sigismember*( ) function shall test whether the signal specified by *signo* is a member of the set pointed to by *set*.

Applications should call either *sigemptyset*( ) or *sigfillset*( ) at least once for each object of type **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is nonetheless supplied as an argument to any of *pthread_sigmask*( ), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigismember*( ), *sigpending*( ), *sigprocmask*( ), *sigsuspend*( ), *sigtimedwait*( ), *sigwait*( ), or *sigwaitinfo*( ), the results are undefined.

RETURN VALUE

Upon successful completion, *sigismember*( ) shall return 1 if the specified signal is a member of the specified set, or 0 if it is not. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *sigismember*( ) function may fail if:

[EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal number.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 513), *pthread_sigmask*( ), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigfillset*( ), *sigemptyset*( ), *sigpending*( ), *sigsuspend*( )

XBD **<signal.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in previous issues.

Issue 6

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

NAME

siglongjmp — non-local goto with signal handling

SYNOPSIS

CX
```
#include <setjmp.h>

_Noreturn void siglongjmp(sigjmp_buf env, int val);
```

DESCRIPTION

The *siglongjmp*( ) function shall be equivalent to the *longjmp*( ) function, except as follows:

- References to *setjmp*( ) shall be equivalent to *sigsetjmp*( ).
- The *siglongjmp*( ) function shall restore the saved signal mask if and only if the *env* argument was initialized by a call to *sigsetjmp*( ) with a non-zero *savemask* argument.

RETURN VALUE

After *siglongjmp*( ) is completed, thread execution shall continue as if the corresponding invocation of *sigsetjmp*( ) had just returned the value specified by *val*. The *siglongjmp*( ) function shall not cause *sigsetjmp*( ) to return 0; if *val* is 0, *sigsetjmp*( ) shall return the value 1.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The distinction between *setjmp*( ) or *longjmp*( ) and *sigsetjmp*( ) or *siglongjmp*( ) is only significant for programs which use *sigaction*( ), *sigprocmask*( ), or *sigsuspend*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

longjmp( ), *pthread_sigmask*( ), *setjmp*( ), *sigsetjmp*( ), *sigsuspend*( )

XBD **<setjmp.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the ISO POSIX-1 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The DESCRIPTION is rewritten in terms of *longjmp*( ).

The SYNOPSIS is marked CX since the presence of this function in the **<setjmp.h>** header is an extension over the ISO C standard.

Issue 8

Austin Group Defect 1302 is applied, adding `_Noreturn` to the SYNOPSIS.

NAME

signal — signal management

SYNOPSIS

```
#include <signal.h>

void (*signal(int sig, void (*func)(int)))(int);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *signal*( ) function chooses one of three ways in which receipt of the signal number *sig* is to be subsequently handled. If the value of *func* is SIG_DFL, default handling for that signal shall occur. If the value of *func* is SIG_IGN, the signal shall be ignored. Otherwise, the application shall ensure that *func* points to a function to be called when that signal occurs. An invocation of such a function because of a signal, or (recursively) of any further functions called by that invocation (other than functions in the standard library), is called a ``signal handler''.

When a signal occurs, and *func* points to a function, it is implementation-defined whether the equivalent of a:

```
signal(sig, SIG_DFL);
```

is executed or the implementation prevents some implementation-defined set of signals (at least including *sig*) from occurring until the current signal handling has completed. (If the value of *sig* is SIGILL, the implementation may alternatively define that no action is taken.) Next the equivalent of:

```
(*func)(sig);
```

is executed. If and when the function returns, if the value of *sig* was SIGFPE, SIGILL, or SIGSEGV or any other implementation-defined value corresponding to a computational exception, the behavior is undefined. Otherwise, the program shall resume execution at the point it was interrupted. The ISO C standard places a restriction on applications relating to the CX use of *raise*( ) from signal handlers. This restriction does not apply to POSIX applications, as POSIX.1-2024 requires *raise*( ) to be async-signal-safe (see Section 2.4.3, on page 516).

CX If the process is multi-threaded, or if the process is single-threaded and a signal handler is executed other than as the result of:

- CX The process calling *abort*( ), *raise*( ), *kill*( ), *pthread_kill*( ), or *sigqueue*( ) to generate a signal that is not blocked
- CX A pending signal being unblocked and being delivered before the call that unblocked it returns

the behavior is undefined if:

- CX The signal handler refers to any object other than *errno* with static or thread storage CX duration that is not a lock-free atomic object, and not a non-modifiable object (for example, string literals, objects that were defined with a const-qualified type, and objects in memory that is mapped read-only), other than by assigning a value to an object CX declared as **volatile sig_atomic_t**, unless the previous modification (if any) to the object happens before the signal handler is called and the return from the signal handler happens before the next modification (if any) to the object.

67627 CX • The signal handler calls any function defined in this standard other than one of the
67628 functions listed in Section 2.4 (on page 513).

67629 At program start-up, the equivalent of:

67630 `signal(sig, SIG_IGN);`

67631 is executed for some signals, and the equivalent of:

67632 `signal(sig, SIG_DFL);`

67633 CX is executed for all other signals (see *exec*).

67634 The *signal*( ) function shall not change the setting of *errno* if successful.

67635 CX The *signal*( ) function is required to be thread-safe. (See Section 2.9.1 (on page 537).)

67636 **RETURN VALUE**

67637 If the request can be honored, *signal*( ) shall return the value of *func* for the most recent call to
67638 *signal*( ) for the specified signal *sig*. Otherwise, SIG_ERR shall be returned and a positive value
67639 shall be stored in *errno*.

67640 **ERRORS**

67641 The *signal*( ) function shall fail if:

67642 CX [EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a
67643 signal that cannot be caught or ignore a signal that cannot be ignored.

67644 The *signal*( ) function may fail if:

67645 CX [EINVAL] An attempt was made to set the action to SIG_DFL for a signal that cannot be
67646 caught or ignored (or both).

67647 **EXAMPLES**

67648 None.

67649 **APPLICATION USAGE**

67650 The *sigaction*( ) function provides a more comprehensive and reliable mechanism for controlling
67651 signals; new applications should use *sigaction*( ) rather than *signal*( ).

67652 **RATIONALE**

67653 The ISO C standard says that the use of *signal*( ) in a multi-threaded program results in
67654 undefined behavior. However, POSIX.1 has required *signal*( ) to be thread-safe since before
67655 threads were added to the ISO C standard.

67656 **FUTURE DIRECTIONS**

67657 None.

67658 **SEE ALSO**

67659 Section 2.4 (on page 513), *exec*, *pause*( ), *raise*( ), *sigaction*( ), *sigsuspend*( ), *waitid*( )

67660 XBD **<signal.h>**

67661 **CHANGE HISTORY**

67662 First released in Issue 1. Derived from Issue 1 of the SVID.

67663 **Issue 5**

67664 Moved from X/OPEN UNIX extension to BASE.

67665 The DESCRIPTION is updated to indicate that the *sigpause*( ) function restores the signal mask of
67666 the process to its original state before returning.

67667 The RETURN VALUE section is updated to indicate that the *sigpause*( ) function suspends

execution of the process until a signal is received, whereupon it returns –1 and sets *errno* to [EINTR].

Issue 6

Extensions beyond the ISO C standard are marked.

The normative text is updated to avoid use of the term ``must'' for application requirements.

The DESCRIPTION is updated for alignment with the ISO/IEC 9899: 1999 standard.

References to the *wait3*( ) function are removed.

The *sighold*( ), *sigignore*( ), *sigrelse*( ), and *sigset*( ) functions are split out onto their own reference page.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0580 [275], XSH/TC1-2008/0581 [66], and XSH/TC1-2008/0582 [105] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0331 [785] is applied.

Issue 8

Austin Group Defect 728 is applied, reducing the set of circumstances in which undefined behavior results when a signal handler refers to an object with static or thread storage duration.

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899: 2018 standard.

NAME

signbit — test sign

SYNOPSIS

```
#include <math.h>

int signbit(real-floating x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *signbit*( ) macro shall determine whether the sign of its argument value is negative. NaNs, zeros, and infinities have a sign bit.

RETURN VALUE

The *signbit*( ) macro shall return a non-zero value if and only if the sign of its argument value is negative.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fpclassify( ), *isfinite*( ), *isinf*( ), *isnan*( ), *isnormal*( )

XBD **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

67715 **NAME**

67716 signgam — log gamma function

67717 **SYNOPSIS**

67718 XSI
```
#include <math.h>

extern int signgam;
```
67719

67720 **DESCRIPTION**

67721 Refer to *lgamma( )*.

NAME

sigpending — examine pending signals

SYNOPSIS

CX
```
#include <signal.h>

int sigpending(sigset_t *set);
```

DESCRIPTION

The *sigpending*( ) function shall store, in the location referenced by the *set* argument, the set of signals that are blocked from delivery to the calling thread and that are pending on the process or the calling thread.

RETURN VALUE

Upon successful completion, *sigpending*( ) shall return 0; otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *pthread_sigmask*( ), *sigaddset*( ), *sigdelset*( ), *sigemptyset*( ), *sigfillset*( ), *sigismember*( )

XBD **<signal.h>**

CHANGE HISTORY

First released in Issue 3.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

67754 **NAME**

67755 sigprocmask — examine and change blocked signals

67756 **SYNOPSIS**

67757 CX
```
#include <signal.h>

int sigprocmask(int how, const sigset_t *restrict set,
    sigset_t *restrict oset);
```

67760 **DESCRIPTION**

67761 Refer to *pthread_sigmask( )*.

NAME

sigqueue — queue a signal to a process

SYNOPSIS

CX
```
#include <signal.h>

int sigqueue(pid_t pid, int signo, union sigval value);
```

DESCRIPTION

The *sigqueue*( ) function shall cause the signal specified by *signo* to be sent with the value specified by *value* to the process specified by *pid*. If *signo* is zero (the null signal), error checking is performed but no signal is actually sent. The null signal can be used to check the validity of *pid*.

The conditions required for a process to have permission to queue a signal to another process are the same as for the *kill*( ) function.

The *sigqueue*( ) function shall return immediately. If SA_SIGINFO is set for *signo* and if the resources were available to queue the signal, the signal shall be queued and sent to the receiving process. If SA_SIGINFO is not set for *signo*, then *signo* shall be sent at least once to the receiving process; it is unspecified whether *value* shall be sent to the receiving process as a result of this call.

If the value of *pid* causes *signo* to be generated for the sending process, and if *signo* is not blocked for the calling thread and if no other thread has *signo* unblocked or is waiting in a *sigwait*( ) function for *signo*, either *signo* or at least the pending, unblocked signal shall be delivered to the calling thread before the *sigqueue*( ) function returns. Should any multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected for delivery, it shall be the lowest numbered one. The selection order between realtime and non-realtime signals, or between multiple pending non-realtime signals, is unspecified.

RETURN VALUE

Upon successful completion, the specified signal shall have been queued, and the *sigqueue*( ) function shall return a value of zero. Otherwise, the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *sigqueue*( ) function shall fail if:

[EAGAIN] No resources are available to queue the signal. The process has already queued {SIGQUEUE_MAX} signals that are still pending at the receiver(s), or a system-wide resource limit has been exceeded.

[EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.

[EPERM] The process does not have appropriate privileges to send the signal to the receiving process.

[ESRCH] The process *pid* does not exist.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *sigqueue*( ) function allows an application to queue a realtime signal to itself or to another process, specifying the application-defined value. This is common practice in realtime applications on existing realtime systems. It was felt that specifying another function in the *sig*... name space already carved out for signals was preferable to extending the interface to *kill*( ).

Such a function became necessary when the put/get event function of the message queues was removed. It should be noted that the *sigqueue*( ) function implies reduced performance in a security-conscious implementation as the access permissions between the sender and receiver have to be checked on each send when the *pid* is resolved into a target process. Such access checks were necessary only at message queue open in the previous interface.

The standard developers required that *sigqueue*( ) have the same semantics with respect to the null signal as *kill*( ), and that the same permission checking be used. But because of the difficulty of implementing the ``broadcast'' semantic of *kill*( ) (for example, to process groups) and the interaction with resource allocation, this semantic was not adopted. The *sigqueue*( ) function queues a signal to a single process specified by the *pid* argument.

The *sigqueue*( ) function can fail if the system has insufficient resources to queue the signal. An explicit limit on the number of queued signals that a process could send was introduced. While the limit is ``per-sender'', this volume of POSIX.1-2024 does not specify that the resources be part of the state of the sender. This would require either that the sender be maintained after exit until all signals that it had sent to other processes were handled or that all such signals that had not yet been acted upon be removed from the queue(s) of the receivers. This volume of POSIX.1-2024 does not preclude this behavior, but an implementation that allocated queuing resources from a system-wide pool (with per-sender limits) and that leaves queued signals pending after the sender exits is also permitted.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.8.1 (on page 528)

XBD **<signal.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 6

The *sigqueue*( ) function is marked as part of the Realtime Signals Extension option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Realtime Signals Extension option.

Issue 7

The *sigqueue*( ) function is moved from the Realtime Signals Extension option to the Base.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0332 [844] is applied.

NAME

sigsetjmp — set jump point for a non-local goto

SYNOPSIS

CX
```
#include <setjmp.h>

int sigsetjmp(sigjmp_buf env, int savemask);
```

DESCRIPTION

The *sigsetjmp*( ) function shall be equivalent to the *setjmp*( ) function, except as follows:

- References to *setjmp*( ) are equivalent to *sigsetjmp*( ).
- References to *longjmp*( ) are equivalent to *siglongjmp*( ).
- If the value of the *savemask* argument is not 0, *sigsetjmp*( ) shall also save the current signal mask of the calling thread as part of the calling environment.

RETURN VALUE

If the return is from a successful direct invocation, *sigsetjmp*( ) shall return 0. If the return is from a call to *siglongjmp*( ), *sigsetjmp*( ) shall return a non-zero value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The distinction between *setjmp*( )/*longjmp*( ) and *sigsetjmp*( )/*siglongjmp*( ) is only significant for programs which use *sigaction*( ), *sigprocmask*( ), or *sigsuspend*( ).

Note that since this function is defined in terms of *setjmp*( ), if *savemask* is zero, it is unspecified whether the signal mask is saved.

RATIONALE

The ISO C standard specifies various restrictions on the usage of the *setjmp*( ) macro in order to permit implementors to recognize the name in the compiler and not implement an actual function. These same restrictions apply to the *sigsetjmp*( ) macro.

There are processors that cannot easily support these calls, but this was not considered a sufficient reason to exclude them.

4.2 BSD and 4.3 BSD provided functions named *_setjmp*( ) and *_longjmp*( ) that, together with *setjmp*( ) and *longjmp*( ), provided the same functionality as *sigsetjmp*( ) and *siglongjmp*( ). On those systems, *setjmp*( ) and *longjmp*( ) saved and restored signal masks, while *_setjmp*( ) and *_longjmp*( ) did not. On System V Release 3 and in corresponding issues of the SVID, *setjmp*( ) and *longjmp*( ) were explicitly defined not to save and restore signal masks. In order to permit existing practice in both cases, the relation of *setjmp*( ) and *longjmp*( ) to signal masks is not specified, and a new set of functions is defined instead.

The *longjmp*( ) and *siglongjmp*( ) functions operate as in the previous issue provided the matching *setjmp*( ) or *sigsetjmp*( ) has been performed in the same thread. Non-local jumps into contexts saved by other threads would be at best a questionable practice and were not considered worthy of standardization.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_sigmask( ), *siglongjmp*( ), *signal*( ), *sigsuspend*( )

XBD **<setjmp.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The DESCRIPTION is reworded in terms of *setjmp*( ).

The SYNOPSIS is marked CX since the presence of this function in the **<setjmp.h>** header is an extension over the ISO C standard.

Issue 8

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

NAME

sigsuspend — wait for a signal

SYNOPSIS

CX
```
#include <signal.h>

int sigsuspend(const sigset_t *sigmask);
```

DESCRIPTION

The *sigsuspend*( ) function shall atomically both replace the current signal mask of the calling thread with the set of signals pointed to by *sigmask* and suspend the thread until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process. This shall not cause any other signals that may have been pending on the process to become pending on the thread.

If the action is to terminate the process then *sigsuspend*( ) shall never return. If the action is to execute a signal-catching function, then *sigsuspend*( ) shall return after the signal-catching function returns, with the signal mask restored to the set that existed prior to the *sigsuspend*( ) call.

It is not possible to block signals that cannot be ignored. This is enforced by the system without causing an error to be indicated.

RETURN VALUE

Since *sigsuspend*( ) suspends thread execution indefinitely, there is no successful completion return value. If a return occurs, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *sigsuspend*( ) function shall fail if:

[EINTR] A signal is caught by the calling process and control is returned from the signal-catching function.

EXAMPLES

None.

APPLICATION USAGE

Normally, at the beginning of a critical code section, a specified set of signals is blocked using the *sigprocmask*( ) function. When the thread has completed the critical section and needs to wait for the previously blocked signal(s), it pauses by calling *sigsuspend*( ) with the mask that was returned by the *sigprocmask*( ) call.

RATIONALE

Code which wants to avoid the ambiguity of the signal mask for thread cancellation handlers can install an additional cancellation handler which resets the signal mask to the expected value.

```
void cleanup(void *arg)
{
    sigset_t *ss = (sigset_t *) arg;
    pthread_sigmask(SIG_SETMASK, ss, NULL);
}

int call_sigsuspend(const sigset_t *mask)
{
    sigset_t oldmask;
    int result;
    pthread_sigmask(SIG_SETMASK, NULL, &oldmask);
    pthread_cleanup_push(cleanup, &oldmask);
```



```
            result = sigsuspend(mask);
            pthread_cleanup_pop(0);
            return result;
        }
```

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 513), *pause( )*, *sigaction( )*, *sigaddset( )*, *sigdelset( )*, *sigemptyset( )*, *sigfillset( )*

XBD **<signal.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The text in the RETURN VALUE section has been changed from ``suspends process execution'' to ``suspends thread execution''. This reflects IEEE PASC Interpretation 1003.1c #40.

Text in the APPLICATION USAGE section has been replaced.

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

Issue 7

SD5-XSH-ERN-122 is applied, adding the example code in the RATIONALE.

Issue 8

Austin Group Defect 1201 is applied, clarifying the atomicity requirements for *sigsuspend( )*.

Austin Group Defect 1223 is applied, changing the example code in the RATIONALE.

NAME

sigtimedwait, sigwaitinfo — wait for queued signals

SYNOPSIS

CX
```
#include <signal.h>

int sigtimedwait(const sigset_t *restrict set,
    siginfo_t *restrict info,
    const struct timespec *restrict timeout);
int sigwaitinfo(const sigset_t *restrict set,
    siginfo_t *restrict info);
```

DESCRIPTION

The *sigtimedwait*( ) function shall be equivalent to *sigwaitinfo*( ) except that if none of the signals specified by *set* are pending, *sigtimedwait*( ) shall wait for the time interval specified in the **timespec** structure referenced by *timeout*. If the **timespec** structure pointed to by *timeout* is zero-valued and if none of the signals specified by *set* are pending, then *sigtimedwait*( ) shall return immediately with an error. If *timeout* is the null pointer, the behavior is unspecified. The CLOCK_MONOTONIC clock shall be used to measure the time interval specified by the *timeout* argument.

The *sigwaitinfo*( ) function selects the pending signal from the set specified by *set*. Should any of multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it shall be the lowest numbered one. The selection order between realtime and non-realtime signals, or between multiple pending non-realtime signals, is unspecified. If no signal in *set* is pending at the time of the call, the calling thread shall be suspended until one or more signals in *set* become pending or until it is interrupted by an unblocked, caught signal.

The *sigwaitinfo*( ) function shall be equivalent to the *sigwait*( ) function, except that the return value and the error reporting method are different (see RETURN VALUE), and that if the *info* argument is non-NULL, the selected signal number shall be stored in the *si_signo* member, and the cause of the signal shall be stored in the *si_code* member. If any value is queued to the selected signal, the first such queued value shall be dequeued and, if the *info* argument is non-NULL, the value shall be stored in the *si_value* member of *info*. The system resource used to queue the signal shall be released and returned to the system for other use. If no value is queued, the content of the *si_value* member is undefined. If no further signals are queued for the selected signal, the pending indication for that signal shall be reset.

RETURN VALUE

Upon successful completion (that is, one of the signals specified by *set* is pending or is generated) *sigwaitinfo*( ) and *sigtimedwait*( ) shall return the selected signal number. Otherwise, the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *sigtimedwait*( ) function shall fail if:

[EAGAIN] No signal specified by *set* was generated within the specified timeout period.

The *sigtimedwait*( ) and *sigwaitinfo*( ) functions may fail if:

[EINTR] The wait was interrupted by an unblocked, caught signal. It shall be documented in system documentation whether this error causes these functions to fail.

The *sigtimedwait*( ) function may also fail if:

[EINVAL] The *timeout* argument specified a *tv_nsec* value less than zero or greater than or equal to 1 000 million.

An implementation should only check for this error if no signal is pending in *set* and it is necessary to wait.

EXAMPLES

None.

APPLICATION USAGE

The *sigtimedwait*( ) function times out and returns an [EAGAIN] error. Application developers should note that this is inconsistent with other functions such as *pthread_cond_timedwait*( ) that return [ETIMEDOUT].

Note that in order to ensure that generated signals are queued and signal values passed to *sigqueue*( ) are available in *si_value*, applications which use *sigwaitinfo*( ) or *sigtimedwait*( ) need to set the SA_SIGINFO flag for each signal in the set (see Section 2.4, on page 513). This means setting each signal to be handled by a three-argument signal-catching function, even if the handler will never be called. It is not possible (portably) to set a signal handler to SIG_DFL while setting the SA_SIGINFO flag, because assigning to the *sa_handler* member of **struct sigaction** instead of the *sa_sigaction* member would result in undefined behavior, and SIG_DFL need not be assignment-compatible with *sa_sigaction*. Even if an assignment of SIG_DFL to *sa_sigaction* is accepted by the compiler, the implementation need not treat this value as special—it could just be taken as the address of a signal-catching function.

RATIONALE

Existing programming practice on realtime systems uses the ability to pause waiting for a selected set of events and handle the first event that occurs in-line instead of in a signal-handling function. This allows applications to be written in an event-directed style similar to a state machine. This style of programming is useful for largescale transaction processing in which the overall throughput of an application and the ability to clearly track states are more important than the ability to minimize the response time of individual event handling.

It is possible to construct a signal-waiting macro function out of the realtime signal function mechanism defined in this volume of POSIX.1-2024. However, such a macro has to include the definition of a generalized handler for all signals to be waited on. A significant portion of the overhead of handler processing can be avoided if the signal-waiting function is provided by the kernel. This volume of POSIX.1-2024 therefore provides two signal-waiting functions—one that waits indefinitely and one with a timeout—as part of the overall realtime signal function specification.

The specification of a function with a timeout allows an application to be written that can be broken out of a wait after a set period of time if no event has occurred. It was argued that setting a timer event before the wait and recognizing the timer event in the wait would also implement the same functionality, but at a lower performance level. Because of the performance degradation associated with the user-level specification of a timer event and the subsequent cancellation of that timer event after the wait completes for a valid event, and the complexity associated with handling potential race conditions associated with the user-level method, the separate function has been included.

Note that the semantics of the *sigwaitinfo*( ) function are nearly identical to that of the *sigwait*( ) function defined by this volume of POSIX.1-2024. The only difference is that *sigwaitinfo*( ) returns the queued signal value in the *value* argument. The return of the queued value is required so that applications can differentiate between multiple events queued to the same signal number.

The two distinct functions are being maintained because some implementations may choose to implement the POSIX Threads Extension functions and not implement the queued signals extensions. Note, though, that *sigwaitinfo*( ) does not return the queued value if the *value* argument is NULL, so the POSIX Threads Extension *sigwait*( ) function can be implemented as a macro on *sigwaitinfo*( ).

The *sigtimedwait*( ) function was separated from the *sigwaitinfo*( ) function to address concerns regarding the overloading of the *timeout* pointer to indicate indefinite wait (no timeout), timed wait, and immediate return, and concerns regarding consistency with other functions where the conditional and timed waits were separate functions from the pure blocking function. The semantics of *sigtimedwait*( ) are specified such that *sigwaitinfo*( ) could be implemented as a macro with a null pointer for *timeout*.

The *sigwait* functions provide a synchronous mechanism for threads to wait for asynchronously-generated signals. One important question was how many threads that are suspended in a call to a *sigwait*( ) function for a signal should return from the call when the signal is sent. Four choices were considered:

1. Return an error for multiple simultaneous calls to *sigwait* functions for the same signal.
2. One or more threads return.
3. All waiting threads return.
4. Exactly one thread returns.

Prohibiting multiple calls to *sigwait*( ) for the same signal was felt to be overly restrictive. The ``one or more'' behavior made implementation of conforming packages easy at the expense of forcing POSIX threads clients to protect against multiple simultaneous calls to *sigwait*( ) in application code in order to achieve predictable behavior. There was concern that the ``all waiting threads'' behavior would result in ``signal broadcast storms'', consuming excessive CPU resources by replicating the signals in the general case. Furthermore, no convincing examples could be presented that delivery to all was either simpler or more powerful than delivery to one.

Thus, the consensus was that exactly one thread that was suspended in a call to a *sigwait* function for a signal should return when that signal occurs. This is not an onerous restriction as:

- A multi-way signal wait can be built from the single-way wait.
- Signals should only be handled by application-level code, as library routines cannot guess what the application wants to do with signals generated for the entire process.
- Applications can thus arrange for a single thread to wait for any given signal and call any needed routines upon its arrival.

In an application that is using signals for interprocess communication, signal processing is typically done in one place. Alternatively, if the signal is being caught so that process cleanup can be done, the signal handler thread can call separate process cleanup routines for each portion of the application. Since the application main line started each portion of the application, it is at the right abstraction level to tell each portion of the application to clean up.

Certainly, there exist programming styles where it is logical to consider waiting for a single signal in multiple threads. A simple *sigwait_multiple*( ) routine can be constructed to achieve this goal. A possible implementation would be to have each *sigwait_multiple*( ) caller registered as having expressed interest in a set of signals. The caller then waits on a thread-specific condition variable. A single server thread calls a *sigwait*( ) function on the union of all registered signals. When the *sigwait*( ) function returns, the appropriate state is set and condition variables are broadcast. New *sigwait_multiple*( ) callers may cause the pending *sigwait*( ) call to be canceled

and reissued in order to update the set of signals being waited for.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 513), Section 2.8.1 (on page 528), *pause*( ), *pthread_sigmask*( ), *sigaction*( ), *sigpending*( ), *sigsuspend*( ), *sigwait*( )

XBD **<signal.h>**, **<time.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 6

These functions are marked as part of the Realtime Signals Extension option.

The Open Group Corrigendum U035/3 is applied. The SYNOPSIS of the *sigwaitinfo*( ) function has been corrected so that the second argument is of type **siginfo_t** *.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Realtime Signals Extension option.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the CLOCK_MONOTONIC clock, if supported, is used to measure timeout intervals.

The **restrict** keyword is added to the *sigtimedwait*( ) and *sigwaitinfo*( ) prototypes for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/130 is applied, restoring wording in the RETURN VALUE section to that in the original base document (``An implementation should only check for this error if no signal is pending in *set* and it is necessary to wait'').

Issue 7

The *sigtimedwait*( ) and *sigwaitinfo*( ) functions are moved from the Realtime Signals Extension option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0583 [392] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0333 [815] is applied.

Issue 8

Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.

NAME

sigwait — wait for queued signals

SYNOPSIS

CX
```
#include <signal.h>

int sigwait(const sigset_t *restrict set, int *restrict sig);
```

DESCRIPTION

The *sigwait*( ) function shall select a pending signal from *set*, atomically clear it from the system's set of pending signals, and return that signal number in the location referenced by *sig*. If prior to the call to *sigwait*( ) there are multiple pending instances of a single signal number, it is implementation-defined whether upon successful return there are any remaining pending signals for that signal number. If the implementation supports queued signals and there are multiple signals queued for the signal number selected, the first such queued signal shall cause a return from *sigwait*( ) and the remainder shall remain queued. If no signal in *set* is pending at the time of the call, the thread shall be suspended until one or more becomes pending. The signals defined by *set* shall have been blocked at the time of the call to *sigwait*( ); otherwise, the behavior is undefined. The effect of *sigwait*( ) on the signal actions for the signals in *set* is unspecified.

If more than one thread is using *sigwait*( ) to wait for the same signal, no more than one of these threads shall return from *sigwait*( ) with the signal number. If more than a single thread is blocked in *sigwait*( ) for a signal when that signal is generated for the process, it is unspecified which of the waiting threads returns from *sigwait*( ). If the signal is generated for a specific thread, as by *pthread_kill*( ), only that thread shall return.

Should any of the multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it shall be the lowest numbered one. The selection order between realtime and non-realtime signals, or between multiple pending non-realtime signals, is unspecified.

RETURN VALUE

Upon successful completion, *sigwait*( ) shall store the signal number of the received signal at the location referenced by *sig* and return zero. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *sigwait*( ) function may fail if:

[EINVAL] The *set* argument contains an invalid or unsupported signal number.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

To provide a convenient way for a thread to wait for a signal, this volume of POSIX.1-2024 provides the *sigwait*( ) function. For most cases where a thread has to wait for a signal, the *sigwait*( ) function should be quite convenient, efficient, and adequate.

However, requests were made for a lower-level primitive than *sigwait*( ) and for semaphores that could be used by threads. After some consideration, threads were allowed to use semaphores and *sem_post*( ) was defined to be async-signal-safe.

In summary, when it is necessary for code run in response to an asynchronous signal to notify a thread, *sigwait*( ) should be used to handle the signal. Alternatively, if the implementation provides semaphores, they also can be used, either following *sigwait*( ) or from within a signal

handling routine previously registered with *sigaction*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 513), Section 2.8.1 (on page 528), *pause*( ), *pthread_sigmask*( ), *sigaction*( ), *sigpending*( ), *sigsuspend*( ), *sigtimedwait*( )

XBD **<signal.h>**, **<time.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 6

The **restrict** keyword is added to the *sigwait*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/131 is applied, updating the DESCRIPTION to state that if more than a single thread is blocked in *sigwait*( ), it is unspecified which of the waiting threads returns, and that if a signal is generated for a specific thread only that thread shall return.

Issue 7

Functionality relating to the Realtime Signals Extension option is moved to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0584 [76] is applied.

68197 **NAME**

68198 sigwaitinfo — wait for queued signals

68199 **SYNOPSIS**

68200 ```
#include <signal.h>
```

68201 ```
int sigwaitinfo(const sigset_t *restrict set, siginfo_t *restrict info);
```

68202 **DESCRIPTION**

68203 Refer to *sigtimedwait( )*.

NAME

sin, sinf, sinl — sine function

SYNOPSIS

```
#include <math.h>

double sin(double x);
float sinf(float x);
long double sinl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the sine of their argument *x*, measured in radians.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the sine of *x*.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is ±Inf, a domain error shall occur, and a NaN shall be returned.

MXX If *x* is subnormal, *x* should be returned.

MX If *x* is subnormal, a range error may occur and, if *x* is not returned, *sin*( ), *sinf*( ), and *sinl*( ) shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

ERRORS

These functions shall fail if:

MX

Domain Error	The *x* argument is ±Inf. If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

MX

Range Error	The value of *x* is subnormal If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

Taking the Sine of a 45-Degree Angle

```
#include <math.h>
...
double radians = 45.0 * M_PI / 180;
double result;
...
result = sin(radians);
```

APPLICATION USAGE

These functions may lose accuracy when their argument is near a multiple of π or is far from 0.0.

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asin( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes in previous issues.

Issue 6

The *sinf*( ) and *sinl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0585 [68] and XSH/TC1-2008/0586 [320] are applied.

Issue 8

Austin Group Defect 1382 is applied, rearranging the text describing the behavior when x is subnormal to avoid the need for two shading changes.

NAME

sinh, sinhf, sinhl — hyperbolic sine functions

SYNOPSIS

```
#include <math.h>

double sinh(double x);
float sinhf(float x);
long double sinhl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the hyperbolic sine of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the hyperbolic sine of *x*.

If the result would cause an overflow, a range error shall occur and ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (with the same sign as *x*) shall be returned as appropriate for the type of the function.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

MXX If *x* is subnormal, *x* should be returned.

MX If *x* is subnormal, a range error may occur and, if *x* is not returned, *sinh*( ), *sinhf*( ), and *sinhl*( ) shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

ERRORS

These functions shall fail if:

Range Error — The result would cause an overflow.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

MX Range Error — The value *x* is subnormal.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asinh( ), *cosh*( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *tanh*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *sinhf*( ) and *sinhl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0587 [68] is applied.

Issue 8

Austin Group Defect 1382 is applied, rearranging the text describing the behavior when *x* is subnormal to avoid the need for two shading changes.

68346 **NAME**

68347 sinl — sine function

68348 **SYNOPSIS**

68349 `#include <math.h>`

68350 `long double sinl(long double x);`

68351 **DESCRIPTION**

68352 Refer to *sin( )*.

68353 **NAME**

68354 sleep — suspend execution for an interval of time

68355 **SYNOPSIS**

68356 `#include <unistd.h>`

68357 `unsigned sleep(unsigned seconds);`

68358 **DESCRIPTION**

68359 The *sleep*( ) function shall cause the calling thread to be suspended from execution until either
68360 the number of realtime seconds specified by the argument *seconds* has elapsed or a signal is
68361 delivered to the calling thread and its action is to invoke a signal-catching function or to
68362 terminate the process. The suspension time may be longer than requested due to the scheduling
68363 of other activity by the system.

68364 In single-threaded programs, *sleep*( ) may make use of SIGALRM. In multi-threaded programs,
68365 *sleep*( ) shall not make use of SIGALRM and the remainder of this DESCRIPTION does not apply.

68366 If a SIGALRM signal is generated for the calling process during execution of *sleep*( ) and if the
68367 SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether *sleep*( )
68368 returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also
68369 unspecified whether it remains pending after *sleep*( ) returns or it is discarded.

68370 If a SIGALRM signal is generated for the calling process during execution of *sleep*( ), except as a
68371 result of a prior call to *alarm*( ), and if the SIGALRM signal is not being ignored or blocked from
68372 delivery, it is unspecified whether that signal has any effect other than causing *sleep*( ) to return.

68373 If a signal-catching function interrupts *sleep*( ) and examines or changes either the time a
68374 SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or
68375 whether the SIGALRM signal is blocked from delivery, the results are unspecified.

68376 If a signal-catching function interrupts *sleep*( ) and calls *siglongjmp*( ) or *longjmp*( ) to restore an
68377 environment saved prior to the *sleep*( ) call, the action associated with the SIGALRM signal and
68378 the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also
68379 unspecified whether the SIGALRM signal is blocked, unless the signal mask of the process is
68380 restored as part of the environment.

68381 **RETURN VALUE**

68382 If *sleep*( ) returns because the requested time has elapsed, the value returned shall be 0. If *sleep*( )
68383 returns due to delivery of a signal, the return value shall be the ``unslept'' amount (the requested
68384 time minus the time actually slept) in seconds.

68385 **ERRORS**

68386 No errors are defined.

68387 **EXAMPLES**

68388 None.

68389 **APPLICATION USAGE**

68390 None.

68391 **RATIONALE**

68392 There are two general approaches to the implementation of the *sleep*( ) function. One is to use the
68393 *alarm*( ) function to schedule a SIGALRM signal and then suspend the calling thread waiting for
68394 that signal. The other is to implement an independent facility. This volume of POSIX.1-2024
68395 permits either approach in single-threaded programs, but the simple alarm/suspend
68396 implementation is not appropriate for multi-threaded programs.

68397 In order to comply with the requirement that no primitive shall change a process attribute unless

68398 explicitly described by this volume of POSIX.1-2024, an implementation using SIGALRM must
68399 carefully take into account any SIGALRM signal scheduled by previous *alarm*( ) calls, the action
68400 previously established for SIGALRM, and whether SIGALRM was blocked. If a SIGALRM has
68401 been scheduled before the *sleep*( ) would ordinarily complete, the *sleep*( ) must be shortened to
68402 that time and a SIGALRM generated (possibly simulated by direct invocation of the signal-
68403 catching function) before *sleep*( ) returns. If a SIGALRM has been scheduled after the *sleep*( )
68404 would ordinarily complete, it must be rescheduled for the same time before *sleep*( ) returns. The
68405 action and blocking for SIGALRM must be saved and restored.

68406 Historical implementations often implement the SIGALRM-based version using *alarm*( ) and
68407 *pause*( ). One such implementation is prone to infinite hangups, as described in *pause*( ).
68408 Another such implementation uses the C-language *setjmp*( ) and *longjmp*( ) functions to avoid
68409 that window. That implementation introduces a different problem: when the SIGALRM signal
68410 interrupts a signal-catching function installed by the user to catch a different signal, the
68411 *longjmp*( ) aborts that signal-catching function. An implementation based on *sigprocmask*( ),
68412 *alarm*( ), and *sigsuspend*( ) can avoid these problems.

68413 Despite all reasonable care, there are several very subtle, but detectable and unavoidable,
68414 differences between the two types of implementations. These are the cases mentioned in this
68415 volume of POSIX.1-2024 where some other activity relating to SIGALRM takes place, and the
68416 results are stated to be unspecified. All of these cases are sufficiently unusual as not to be of
68417 concern to most applications.

68418 See also the discussion of the term *realtime* in *alarm*( ).

68419 Since *sleep*( ) can be implemented using *alarm*( ), the discussion about alarms occurring early
68420 under *alarm*( ) applies to *sleep*( ) as well.

68421 Application developers should note that the type of the argument *seconds* and the return value of
68422 *sleep*( ) is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces Application
68423 cannot pass a value greater than the minimum guaranteed value for {UINT_MAX}, which the
68424 ISO C standard sets as 65 535, and any application passing a larger value is restricting its
68425 portability. A different type was considered, but historical implementations, including those
68426 with a 16-bit **int** type, consistently use either **unsigned** or **int**.

68427 Scheduling delays may cause the process to return from the *sleep*( ) function significantly after
68428 the requested time. In such cases, the return value should be set to zero, since the formula
68429 (requested time minus the time actually spent) yields a negative number and *sleep*( ) returns an
68430 **unsigned**.

68431 **FUTURE DIRECTIONS**

68432 A future version of this standard may require that *sleep*( ) does not make use of SIGALRM in all
68433 programs, not just multi-threaded programs.

68434 **SEE ALSO**

68435 *alarm*( ), *nanosleep*( ), *pause*( ), *sigaction*( ), *sigsetjmp*( ), *timer_create*( )

68436 XBD **<unistd.h>**

68437 **CHANGE HISTORY**

68438 First released in Issue 1. Derived from Issue 1 of the SVID.

68439 **Issue 5**

68440 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

68441 **Issue 6**

68442 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/132 is applied, making a correction in the
68443 RATIONALE section.

68444 **Issue 7**

68445 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0334 [625] is applied.

68446 **Issue 8**

68447 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

68448 **NAME**

68449 snprintf — print formatted output

68450 **SYNOPSIS**

68451 ```
#include <stdio.h>
```

68452 ```
int snprintf(char *restrict s, size_t n,
```

68453 ```
    const char *restrict format, ...);
```

68454 **DESCRIPTION**

68455 Refer to *fprintf( )*.

  
```

NAME

sockatmark — determine whether a socket is at the out-of-band mark

SYNOPSIS

```
#include <sys/socket.h>

int sockatmark(int s);
```

DESCRIPTION

The *sockatmark*( ) function shall determine whether the socket specified by the descriptor *s* is at the out-of-band data mark (see Section 2.10.12, on page 552). If the protocol for the socket supports out-of-band data by marking the stream with an out-of-band data mark, the *sockatmark*( ) function shall return 1 when all data preceding the mark has been read and the out-of-band data mark is the first element in the receive queue. The *sockatmark*( ) function shall not remove the mark from the stream.

RETURN VALUE

Upon successful completion, the *sockatmark*( ) function shall return a value indicating whether the socket is at an out-of-band data mark. If the protocol has marked the data stream and all data preceding the mark has been read, the return value shall be 1; if there is no mark, or if data precedes the mark in the receive queue, the *sockatmark*( ) function shall return 0. Otherwise, it shall return a value of –1 and set *errno* to indicate the error.

ERRORS

The *sockatmark*( ) function shall fail if:

[EBADF] The *s* argument is not a valid file descriptor.

[ENOTTY] The file associated with the *s* argument is not a socket.

EXAMPLES

None.

APPLICATION USAGE

The use of this function between receive operations allows an application to determine which received data precedes the out-of-band data and which follows the out-of-band data.

There is an inherent race condition in the use of this function. On an empty receive queue, the current read of the location might well be at the ``mark'', but the system has no way of knowing that the next data segment that will arrive from the network will carry the mark, and *sockatmark*( ) will return false, and the next read operation will silently consume the mark.

Hence, this function can only be used reliably when the application already knows that the out-of-band data has been seen by the system or that it is known that there is data waiting to be read at the socket (via SIGURG or *select*( )). See Section 2.10.11 (on page 552), Section 2.10.12 (on page 552), Section 2.10.14 (on page 553), and *pselect*( ) for details.

RATIONALE

The *sockatmark*( ) function replaces the historical SIOCATMARK command to *ioctl*( ) which implemented the same functionality on many implementations. Using a wrapper function follows the adopted conventions to avoid specifying commands to the *ioctl*( ) function. The *sockatmark*( ) function could be implemented as follows:

```
#include <sys/ioctl.h>

int sockatmark(int s)
{
    int val;
    if (ioctl(s,SIOCATMARK,&val)==-1)
```



```
            return(-1);
        return(val);
    }
```

The use of [ENOTTY] to indicate an incorrect descriptor type matches the historical behavior of SIOCATMARK.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.10.12 (on page 552), *pselect( )*, *recv( )*, *recvmsg( )*

XBD **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

Issue 7

SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

Issue 8

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

68517 **NAME**

68518 socket — create an endpoint for communication

68519 **SYNOPSIS**

68520 `#include <sys/socket.h>`

68521 `int socket(int domain, int type, int protocol);`

68522 **DESCRIPTION**

68523 The *socket*( ) function shall create an unbound socket in a communications domain, and return a
68524 file descriptor that can be used in later function calls that operate on sockets. The file descriptor
68525 shall be allocated as described in Section 2.6 (on page 525).

68526 The *socket*( ) function takes the following arguments:

68527 *domain* Specifies the communications domain in which a socket is to be created.

68528 *type* Specifies the type of socket to be created.

68529 *protocol* Specifies a particular protocol to be used with the socket. Specifying a *protocol*
68530 of 0 causes *socket*( ) to use an unspecified default protocol appropriate for the
68531 requested socket type.

68532 The *domain* argument specifies the address family used in the communications domain. The
68533 address families supported by the system are implementation-defined.

68534 Symbolic constants that can be used for the domain argument are defined in the **<sys/socket.h>**
68535 header.

68536 The *type* argument specifies the socket type, which determines the semantics of communication
68537 over the socket. The following socket types are defined; implementations may specify additional
68538 socket types:

68539 SOCK_STREAM Provides sequenced, reliable, bidirectional, connection-mode byte streams,
68540 and may provide a transmission mechanism for out-of-band data.

68541 SOCK_DGRAM Provides datagrams, which are connectionless-mode, unreliable messages of
68542 fixed maximum length.

68543 SOCK_SEQPACKET
68544 Provides sequenced, reliable, bidirectional, connection-mode transmission
68545 paths for records. A record can be sent using one or more output operations
68546 and received using one or more input operations, but a single operation never
68547 transfers part of more than one record. Record boundaries are visible to the
68548 receiver via the MSG_EOR flag.

68549 Additionally, the *type* argument can contain the bitwise-inclusive OR of flags from the following
68550 list:

68551 SOCK_CLOEXEC Atomically set the FD_CLOEXEC flag on the new file descriptor.

68552 SOCK_CLOFORK Atomically set the FD_CLOFORK flag on the new file descriptor.

68553 SOCK_NONBLOCK Set the O_NONBLOCK file status flag on the new file description.

68554 Implementations may define additional flags.

68555 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address
68556 family. If the *protocol* argument is zero, the default protocol for this address family and type shall
68557 be used. The protocols supported by the system are implementation-defined.

68558 The process may need to have appropriate privileges to use the *socket*( ) function or to create

some sockets.

RETURN VALUE

Upon successful completion, *socket*( ) shall return a non-negative integer, the socket file descriptor. Otherwise, a value of −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *socket*( ) function shall fail if:

[EAFNOSUPPORT]
The implementation does not support the specified address family.

[EMFILE] All file descriptors available to the process are currently open.

[ENFILE] No more file descriptors are available for the system.

[EPROTONOSUPPORT]
The value of *protocol* is non-zero and either the protocol is not supported by the address family or the protocol is not supported by the implementation.

[EPROTOTYPE] The value of *protocol* is non-zero and the socket type is not supported by the protocol.

OB [ESOCKTNOSUPPORT] or [EPROTONOSUPPORT] or [EPROTOTYPE]
The socket type is not supported by the address family, or the socket type is not supported by the implementation.

The *socket*( ) function may fail if:

[EACCES] The process does not have appropriate privileges.

[ENOBUFS] Insufficient resources were available in the system to perform the operation.

[ENOMEM] Insufficient memory was available to fulfill the request.

EXAMPLES

None.

APPLICATION USAGE

The documentation for specific address families specifies which protocols each address family supports. The documentation for specific protocols specifies which socket types each protocol supports.

The application can determine whether an address family is supported by trying to create a socket with *domain* set to the protocol in question.

RATIONALE

The use of the SOCK_CLOEXEC and SOCK_CLOFORK flags in the *type* argument of *socket*( ) is necessary to avoid a data race in multi-threaded applications. Without SOCK_CLOFORK, a file descriptor is leaked into a child process created by one thread in the window between another thread calling *socket*( ) and using *fcntl*( ) to set the FD_CLOFORK flag. Without SOCK_CLOEXEC, a file descriptor intentionally inherited by child processes is similarly leaked into an executed program if FD_CLOEXEC is not set atomically. The SOCK_NONBLOCK flag is for convenience in avoiding additional *fcntl*( ) calls.

Historically the standard did not specify the *errno* value to be used when the socket type is not supported, and there were differences between implementations. Some reused the existing standard [EPROTONOSUPPORT] or [EPROTOTYPE] values while others set *errno* to a (then) non-standard value of [ESOCKTNOSUPPORT]. All three values are permitted in this version of the standard, but the use of [EPROTONOSUPPORT] or [EPROTOTYPE] is considered to be misleading when no protocol is specified (that is, the value of *protocol* is zero) and consequently

those alternatives have been marked obsolescent. If *protocol* is non-zero, since there is no precedence between error conditions, all three values will still be permitted even after the obsolescent alternatives for the [ESOCKTNOSUPPORT] condition have been removed.

FUTURE DIRECTIONS

A future version of this standard may disallow setting *errno* to [EPROTONOSUPPORT] or [EPROTOTYPE] when the socket type is not supported and *protocol* is zero.

SEE ALSO

Section 2.6 (on page 525), *accept*( ), *bind*( ), *close*( ), *connect*( ), *getsockname*( ), *getsockopt*( ), *listen*( ), *recv*( ), *recvfrom*( ), *recvmsg*( ), *send*( ), *sendmsg*( ), *setsockopt*( ), *shutdown*( ), *socketpair*( )

XBD **<netinet/in.h>**, **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0335 [835] is applied.

Issue 8

Austin Group Defects 411 and 1318 are applied, adding SOCK_CLOEXEC, SOCK_CLOFORK, and SOCK_NONBLOCK.

Austin Group Defect 1067 is applied, adding the [ESOCKTNOSUPPORT] error and changing the [EPROTONOSUPPORT] and [EPROTOTYPE] errors.

Austin Group Defect 1475 is applied, adding *close*( ) to the SEE ALSO section.

68623 **NAME**

68624 socketpair — create a pair of connected sockets

68625 **SYNOPSIS**

68626 ```
#include <sys/socket.h>
```

68627 ```
int socketpair(int domain, int type, int protocol,
```

68628 ```
    int socket_vector[2]);
```

68629 **DESCRIPTION**

68630 The *socketpair*( ) function shall create an unbound pair of connected sockets in a specified *domain*,
68631 of a specified *type*, under the protocol optionally specified by the *protocol* argument. The two
68632 sockets shall be identical. The file descriptors used in referencing the created sockets shall be
68633 returned in *socket_vector*[0] and *socket_vector*[1]. The file descriptors shall be allocated as
68634 described in Section 2.6 (on page 525).

68635 The *socketpair*( ) function takes the following arguments:

68636 *domain* Specifies the communications domain in which the sockets are to be created.

68637 *type* Specifies the type of sockets to be created.

68638 *protocol* Specifies a particular protocol to be used with the sockets. Specifying a
68639 *protocol* of 0 causes *socketpair*( ) to use an unspecified default protocol
68640 appropriate for the requested socket type.

68641 *socket_vector* Specifies a 2-integer array to hold the file descriptors of the created socket pair.

68642 The *type* argument specifies the socket type, which determines the semantics of communications
68643 over the socket. The following socket types are defined; implementations may specify additional
68644 socket types:

68645 SOCK_STREAM Provides sequenced, reliable, bidirectional, connection-mode byte
68646 streams, and may provide a transmission mechanism for out-of-band
68647 data.

68648 SOCK_DGRAM Provides datagrams, which are connectionless-mode, unreliable messages
68649 of fixed maximum length.

68650 SOCK_SEQPACKET Provides sequenced, reliable, bidirectional, connection-mode transmission
68651 paths for records. A record can be sent using one or more output
68652 operations and received using one or more input operations, but a single
68653 operation never transfers part of more than one record. Record
68654 boundaries are visible to the receiver via the MSG_EOR flag.

68655 Additionally, the *type* argument can contain the bitwise-inclusive OR of flags from the following
68656 list:

68657 SOCK_CLOEXEC Atomically set the FD_CLOEXEC flag on the new file descriptors.

68658 SOCK_CLOFORK Atomically set the FD_CLOFORK flag on the new file descriptors.

68659 SOCK_NONBLOCK Set the O_NONBLOCK file status flag on the new file descriptions.

68660 Implementations may define additional flags.

68661 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address
68662 family. If the *protocol* argument is zero, the default protocol for this address family and type shall
68663 be used. The protocols supported by the system are implementation-defined.

68664 The process may need to have appropriate privileges to use the *socketpair*( ) function or to create
68665 some sockets.

  
```

68666 **RETURN VALUE**

68667 Upon successful completion, this function shall return 0; otherwise, –1 shall be returned and
68668 *errno* set to indicate the error, no file descriptors shall be allocated, and the contents of
68669 *socket_vector* shall be left unmodified.

68670 **ERRORS**

68671 The *socketpair*( ) function shall fail if:

68672 [EAFNOSUPPORT]
68673 The implementation does not support the specified address family.

68674 [EMFILE] All, or all but one, of the file descriptors available to the process are currently
68675 open.

68676 [ENFILE] No more file descriptors are available for the system.

68677 [EOPNOTSUPP] The specified protocol does not permit creation of socket pairs.

68678 [EPROTONOSUPPORT]
68679 The protocol is not supported by the address family, or the protocol is not
68680 supported by the implementation.

68681 [EPROTOTYPE] The socket type is not supported by the protocol.

68682 The *socketpair*( ) function may fail if:

68683 [EACCES] The process does not have appropriate privileges.

68684 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

68685 [ENOMEM] Insufficient memory was available to fulfill the request.

68686 **EXAMPLES**

68687 None.

68688 **APPLICATION USAGE**

68689 The documentation for specific address families specifies which protocols each address family
68690 supports. The documentation for specific protocols specifies which socket types each protocol
68691 supports.

68692 The *socketpair*( ) function is used primarily with UNIX domain sockets and need not be
68693 supported for other domains.

68694 **RATIONALE**

68695 The use of the SOCK_CLOEXEC and SOCK_CLOFORK flags in the *type* argument of *socketpair*( )
68696 is necessary to avoid a data race in multi-threaded applications. Without SOCK_CLOFORK, a
68697 file descriptor is leaked into a child process created by one thread in the window between
68698 another using *socketpair*( ) and using using *fcntl*( ) to set the FD_CLOFORK flag. Without
68699 SOCK_CLOEXEC, a file descriptor intentionally inherited by child processes is similarly leaked
68700 into an executed program if FD_CLOEXEC is not set atomically.

68701 Since socket pairs are often used for communication between a parent and child process,
68702 SOCK_CLOFORK has to be used with care in order for the pair to be usable. If the parent will be
68703 writing and the child will be reading, SOCK_CLOFORK should be used when creating the pair,
68704 and then *fcntl*( ) should be used to clear FD_CLOFORK for the read side of the pair. This
68705 prevents the write side from leaking into other children, ensuring the child will get end-of-file
68706 when the parent closes the write side (although the read side can still be leaked). If the parent
68707 will be reading and the child will be writing, or if the socket pair will be used bidirectionally,
68708 there is no way to prevent the write side(s) being leaked (short of preventing other threads from
68709 creating child processes) in order to ensure the parent gets end-of-file when the child closes its

side, and so the two processes should use an alternative method of indicating the end of communications, for example using *shutdown*( ).

Arranging for FD_CLOEXEC to be set appropriately is more straightforward. The parent should use SOCK_CLOEXEC when creating the socket pair and the child should clear FD_CLOEXEC on the side to be passed to the new program before calling an *exec* family function to execute it.

The SOCK_NONBLOCK flag is for convenience in avoiding additional *fcntl*( ) calls.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.6 (on page 525), *socket*( )

XBD **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7

The description of the [EMFILE] error condition is aligned with the *pipe*( ) function.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0336 [835] and XSH/TC2-2008/0337 [483,835] are applied.

Issue 8

Austin Group Defects 411 and 1318 are applied, adding SOCK_CLOEXEC, SOCK_CLOFORK, and SOCK_NONBLOCK.

68730 **NAME**

68731 sprintf — print formatted output

68732 **SYNOPSIS**

68733 `#include <stdio.h>`

68734 `int sprintf(char *restrict s, const char *restrict format, ...);`

68735 **DESCRIPTION**

68736 Refer to *fprintf( )*.

NAME

sqrt, sqrtf, sqrtl — square root function

SYNOPSIS

```
#include <math.h>

double sqrt(double x);
float sqrtf(float x);
long double sqrtl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the square root of their argument x, $\sqrt{x}$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the square root of x.

MX The returned value shall be dependent on the current rounding direction mode.

MX For finite values of $x < -0$, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

MX If x is NaN, a NaN shall be returned.

If x is ±0 or +Inf, x shall be returned.

If x is –Inf, a domain error shall occur, and a NaN shall be returned.

ERRORS

These functions shall fail if:

MX Domain Error — The finite value of x is < -0, or x is –Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

EXAMPLES

Taking the Square Root of 9.0

```
#include <math.h>
...
double x = 9.0;
double result;
...
result = sqrt(x);
```


APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( )

XBD Section 4.23 (on page 109), **<math.h>**, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *sqrtf*( ) and *sqrtl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0588 [320] is applied.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018 standard.

68802 **NAME**

68803 srand — pseudo-random number generator

68804 **SYNOPSIS**

68805 `#include <stdlib.h>`

68806 `void srand(unsigned seed);`

68807 **DESCRIPTION**

68808 Refer to *rand( )*.

NAME

srand48 — seed the uniformly distributed double-precision pseudo-random number generator

SYNOPSIS

XSI

```
#include <stdlib.h>

void srand48(long seedval);
```

DESCRIPTION

Refer to *drand48( )*.

68816 **NAME**

68817 srandom — seed pseudo-random number generator

68818 **SYNOPSIS**

68819 XSI
```
#include <stdlib.h>

void srandom(unsigned seed);
```

68821 **DESCRIPTION**

68822 Refer to *initstate( )*.

68823 **NAME**

68824 sscanf — convert formatted input

68825 **SYNOPSIS**

68826 `#include <stdio.h>`

68827 `int sscanf(const char *restrict s, const char *restrict format, ...);`

68828 **DESCRIPTION**

68829 Refer to *fscanf( )*.

68830 **NAME**

68831 stat — get file status

68832 **SYNOPSIS**

68833 `#include <sys/stat.h>`

68834 `int stat(const char *restrict path, struct stat *restrict buf);`

68835 **DESCRIPTION**

68836 Refer to *fstatat*( ).

68837 **NAME**

68838 statvfs — get file system information

68839 **SYNOPSIS**

68840 `#include <sys/statvfs.h>`

68841 `int statvfs(const char *restrict path, struct statvfs *restrict buf);`

68842 **DESCRIPTION**

68843 Refer to *fstatvfs( )*.

NAME

stderr, stdin, stdout — standard I/O streams

SYNOPSIS

```
#include <stdio.h>

extern FILE *stderr, *stdin, *stdout;
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type **FILE**. The *fopen*( ) function shall create certain descriptive data for a stream and return a pointer to designate the stream in all further transactions. Normally, there are three open streams with constant pointers declared in the **<stdio.h>** header and associated with the standard open files.

At program start-up, three streams shall be predefined and already open: *stdin* (standard input, for conventional input) for reading, *stdout* (standard output, for conventional output) for writing, and *stderr* (standard error, for diagnostic output) for writing. When opened, *stderr* shall CX not be fully buffered; *stdin* and *stdout* shall be fully buffered if and only if the file descriptor associated with the stream is determined not to be associated with an interactive device.

CX The following symbolic values in **<unistd.h>** define the file descriptors that shall be associated with the C-language *stdin*, *stdout*, and *stderr* when the application is started:

STDIN_FILENO	Standard input value, *stdin*. Its value is 0.
STDOUT_FILENO	Standard output value, *stdout*. Its value is 1.
STDERR_FILENO	Standard error value, *stderr*. Its value is 2.

These file descriptors are often all associated with a single open file description which has access mode O_RDWR (e.g., in the case of a terminal device for a login shell). However, the *stderr*, *stdin*, and *stdout* streams need not be opened for both reading and writing at program start-up in this case.

RETURN VALUE

None.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fclose( ), *feof*( ), *ferror*( ), *fileno*( ), *fopen*( ), *fprintf*( ), *fread*( ), *fscanf*( ), *fseek*( ), *getc*( ), *isatty*( ), *popen*( ), *putc*( ), *puts*( ), *read*( ), *setbuf*( ), *setvbuf*( ), *tmpfile*( ), *ungetc*( ), *vfprintf*( )

XBD **<stdio.h>**, **<unistd.h>**

68887 **CHANGE HISTORY**

68888 First released in Issue 1.

68889 **Issue 6**

68890 Extensions beyond the ISO C standard are marked.

68891 A note that *stderr* is expected to be open for reading and writing is added to the DESCRIPTION.

68892 **Issue 8**

68893 Austin Group Defect 1347 is applied, clarifying the requirements for how *stderr*, *stdin*, and *stdout*
68894 are opened at program start-up.

68895 **NAME**

68896 stpcpy — copy a string and return a pointer to the end of the result

68897 **SYNOPSIS**

68898 CX
```
#include <string.h>

char *stpcpy(char *restrict s1, const char *restrict s2);
```

68900 **DESCRIPTION**

68901 Refer to *strcpy( )*.

68902 **NAME**

68903 stpncpy — copy fixed length string, returning a pointer to the array end

68904 **SYNOPSIS**

68905 CX
```
#include <string.h>
```
68906
```
char *stpncpy(char *restrict s1, const char *restrict s2, size_t size);
```

68907 **DESCRIPTION**

68908 Refer to *strncpy( )*.

68909 **NAME**

68910 str2sig — translate between signal names and numbers

68911 **SYNOPSIS**

68912 CX
```
#include <signal.h>

int str2sig(const char *restrict str, int *restrict pnum);
```
68913

68914 **DESCRIPTION**

68915 Refer to *sig2str( )*.

NAME

strcasecmp, strcasecmp_l, strncasecmp, strncasecmp_l — case-insensitive string comparisons

SYNOPSIS

```
#include <strings.h>

int strcasecmp(const char *s1, const char *s2);
int strcasecmp_l(const char *s1, const char *s2,
    locale_t locale);
int strncasecmp(const char *s1, const char *s2, size_t n);
int strncasecmp_l(const char *s1, const char *s2,
    size_t n, locale_t locale);
```

DESCRIPTION

The *strcasecmp*( ) and *strcasecmp_l*( ) functions shall compare, while ignoring differences in case, the string pointed to by *s1* to the string pointed to by *s2*. The *strncasecmp*( ) and *strncasecmp_l*( ) functions shall compare, while ignoring differences in case, not more than *n* bytes from the string pointed to by *s1* to the string pointed to by *s2*.

The *strcasecmp*( ) and *strncasecmp*( ) functions use the current locale to determine the case of the characters.

The *strcasecmp_l*( ) and *strncasecmp_l*( ) functions use the locale represented by *locale* to determine the case of the characters.

When the *LC_CTYPE* category of the locale being used is from the POSIX locale, these functions shall behave as if the strings had been converted to lowercase and then a byte comparison performed, and *errno* shall not be changed on valid input. Otherwise, the results are unspecified.

The behavior is undefined if the *locale* argument to *strcasecmp_l*( ) or *strncasecmp_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

Upon completion, *strcasecmp*( ) and *strcasecmp_l*( ) shall return an integer greater than, equal to, or less than 0, if the string pointed to by *s1* is, ignoring case, greater than, equal to, or less than the string pointed to by *s2*, respectively.

Upon successful completion, *strncasecmp*( ) and *strncasecmp_l*( ) shall return an integer greater than, equal to, or less than 0, if the possibly null-terminated array pointed to by *s1* is, ignoring case, greater than, equal to, or less than the possibly null-terminated array pointed to by *s2*, respectively.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

68958 **SEE ALSO**

68959 *wcscasecmp*( )

68960 XBD **<strings.h>**

68961 **CHANGE HISTORY**

68962 First released in Issue 4, Version 2.

68963 **Issue 5**

68964 Moved from X/OPEN UNIX extension to BASE.

68965 **Issue 7**

68966 The *strcasecmp*( ) and *strncasecmp*( ) functions are moved from the XSI option to the Base.

68967 The *strcasecmp_l*( ) and *strncasecmp_l*( ) functions are added from The Open Group Technical
68968 Standard, 2006, Extended API Set Part 4.

68969 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0589 [302], XSH/TC1-2008/0590 [294],
68970 XSH/TC1-2008/0591 [283], and XSH/TC1-2008/0592 [283] are applied.

68971 **Issue 8**

68972 Austin Group Defect 448 is applied, adding a requirement that *errno* is not changed on valid
68973 input.

NAME

strcat — concatenate two strings

SYNOPSIS

```
#include <string.h>

char *strcat(char *restrict s1, const char *restrict s2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *strcat*( ) function shall append a copy of the string pointed to by *s2* (including the terminating NUL character) to the end of the string pointed to by *s1*. The initial byte of *s2* overwrites the NUL character at the end of *s1*. If copying takes place between objects that overlap, the behavior is undefined.

CX The *strcat*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

The *strcat*( ) function shall return *s1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

This version is aligned with the ISO C standard; this does not affect compatibility with XPG3 applications. Reliable error detection by this function was never guaranteed.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strncat( )

XBD **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The *strcat*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *strcat*( ) does not change the setting of *errno* on valid input.

NAME

strchr — string scanning operation

SYNOPSIS

```
#include <string.h>

char *strchr(const char *s, int c);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *strchr*( ) function shall locate the first occurrence of *c* (converted to a **char**) in the string pointed to by *s*. The terminating NUL character is considered to be part of the string.

CX The *strchr*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

Upon completion, *strchr*( ) shall return a pointer to the byte, or a null pointer if the byte was not found.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strrchr( )

XBD **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *strchr*( ) does not change the setting of *errno* on valid input.

NAME

strcmp — compare two strings

SYNOPSIS

```
#include <string.h>

int strcmp(const char *s1, const char *s2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *strcmp*( ) function shall compare the string pointed to by *s1* to the string pointed to by *s2*.

The sign of a non-zero return value shall be determined by the sign of the difference between the values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings being compared.

CX The *strcmp*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

Upon completion, *strcmp*( ) shall return an integer greater than, equal to, or less than 0, if the string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2*, respectively.

ERRORS

No errors are defined.

EXAMPLES

Checking a Password Entry

The following example compares the information read from standard input to the value of the name of the user entry. If the *strcmp*( ) function returns 0 (indicating a match), a further check will be made to see if the user entered the proper old password. The *crypt*( ) function shall encrypt the old password entered by the user, using the value of the encrypted password in the **passwd** structure as the salt. If this value matches the value of the encrypted **passwd** in the structure, the entered password *oldpasswd* is the correct user's password. Finally, the program encrypts the new password so that it can store the information in the **passwd** structure.

```
#include <string.h>
#include <unistd.h>
#include <stdio.h>
...
int valid_change;
struct passwd *p;
char user[100];
char oldpasswd[100];
char newpasswd[100];
char savepasswd[100];
...
if (strcmp(p->pw_name, user) == 0) {
    if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
        strcpy(savepasswd, crypt(newpasswd, user));
        p->pw_passwd = savepasswd;
        valid_change = 1;
    }
```



```
            else {
                fprintf(stderr, "Old password is not valid\n");
            }
        }
        ...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strncmp( )

XBD **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *strcmp*( ) does not change the setting of *errno* on valid input.

NAME

strcoll, strcoll_l — string comparison using collating information

SYNOPSIS

```
#include <string.h>

int strcoll(const char *s1, const char *s2);
CX int strcoll_l(const char *s1, const char *s2,
       locale_t locale);
```

DESCRIPTION

CX For *strcoll*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *strcoll*( ) and *strcoll_l*( ) functions shall compare the string pointed to by *s1* to the string pointed to by *s2*, both interpreted as appropriate to the *LC_COLLATE* category of the current CX locale, or of the locale represented by *locale*, respectively.

CX The *strcoll*( ) and *strcoll_l*( ) functions shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error, an application wishing to check for error CX situations should set *errno* to 0, then call *strcoll*( ), or *strcoll_l*( ) then check *errno*.

CX The behavior is undefined if the *locale* argument to *strcoll_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

Upon successful completion, *strcoll*( ) shall return an integer greater than, equal to, or less than 0, according to whether the string pointed to by *s1* is greater than, equal to, or less than the string CX pointed to by *s2* when both are interpreted as appropriate to the current locale. On error, *strcoll*( ) may set *errno*, but no return value is reserved to indicate an error.

Upon successful completion, *strcoll_l*( ) shall return an integer greater than, equal to, or less than 0, according to whether the string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2* when both are interpreted as appropriate to the locale represented by *locale*. On error, *strcoll_l*( ) may set *errno*, but no return value is reserved to indicate an error.

ERRORS

These functions may fail if:

CX [EINVAL] The *s1* or *s2* arguments contain characters outside the domain of the collating sequence.

EXAMPLES

Comparing Nodes

The following example uses an application-defined function, *node_compare*( ), to compare two nodes based on an alphabetical ordering of the *string* field.

```
#include <string.h>
...
struct node { /* These are stored in the table. */
    char *string;
    int length;
};
...
```



```
int node_compare(const void *node1, const void *node2)
{
    return strcoll(((const struct node *)node1)->string,
        ((const struct node *)node2)->string);
}
...
```

APPLICATION USAGE

The *strxfrm*( ) and *strcmp*( ) functions should be used for sorting large lists.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

alphasort( ), *strcmp*( ), *strxfrm*( )

XBD **<string.h>**

CHANGE HISTORY

First released in Issue 3.

Issue 5

The DESCRIPTION is updated to indicate that *errno* does not change if the function is successful.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EINVAL] optional error condition is added.

An example is added.

Issue 7

The *strcoll_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0593 [283] and XSH/TC1-2008/0594 [283] are applied.

NAME

stpcpy, strcpy — copy a string

SYNOPSIS

```
#include <string.h>

char *stpcpy(char *restrict s1, const char *restrict s2);
char *strcpy(char *restrict s1, const char *restrict s2);
```

DESCRIPTION

CX For *strcpy*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *stpcpy*() and *strcpy*() functions shall copy the string pointed to by *s2* (including the terminating NUL character) into the array pointed to by *s1*.

If copying takes place between objects that overlap, the behavior is undefined.

CX The *strcpy*() and *stpcpy*() functions shall not change the setting of *errno* on valid input.

RETURN VALUE

CX The *stpcpy*() function shall return a pointer to the terminating NUL character copied into the *s1* buffer.

The *strcpy*() function shall return *s1*.

No return values are reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

Construction of a Multi-Part Message in a Single Buffer

```
#include <string.h>
#include <stdio.h>

int
main (void)
{
    char buffer [10];
    char *name = buffer;

    name = stpcpy (stpcpy (stpcpy (name, "ice"),"-"), "cream");
    puts (buffer);
    return 0;
}
```

Initializing a String

The following example copies the string `"----------"` into the *permstring* variable.

```
#include <string.h>
...
static char permstring[11];
...
strcpy(permstring, "----------");
...
```


Storing a Key and Data

The following example allocates space for a key using *malloc*( ) then uses *strcpy*( ) to place the key there. Then it allocates space for data using *malloc*( ), and uses *strcpy*( ) to place data there. (The user-defined function *dbfree*( ) frees memory previously allocated to an array of type **struct element** *.)

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
...
/* Structure used to read data and store it. */
struct element {
    char *key;
    char *data;
};

struct element *tbl, *curtbl;
char *key, *data;
int count;
...
void dbfree(struct element *, int);
...
if ((curtbl->key = malloc(strlen(key) + 1)) == NULL) {
    perror("malloc"); dbfree(tbl, count); return NULL;
}
strcpy(curtbl->key, key);

if ((curtbl->data = malloc(strlen(data) + 1)) == NULL) {
    perror("malloc"); free(curtbl->key); dbfree(tbl, count); return NULL;
}
strcpy(curtbl->data, data);
...
```

APPLICATION USAGE

Character movement is performed differently in different implementations. Thus, overlapping moves may yield surprises.

This version is aligned with the ISO C standard; this does not affect compatibility with XPG3 applications. Reliable error detection by this function was never guaranteed.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strncpy( ), *wcscpy*( )

XBD **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

69271 **Issue 6**

69272 The *strcpy*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

69273 **Issue 7**

69274 The *stpcpy*( ) function is added from The Open Group Technical Standard, 2006, Extended API
69275 Set Part 1.

69276 **Issue 8**

69277 Austin Group Defect 448 is applied, adding a requirement that *strcpy*( ) and *stpcpy*( ) do not
69278 change the setting of *errno* on valid input.

69279 Austin Group Defect 1787 is applied, changing the NAME section.

NAME

strcspn — get the length of a complementary substring

SYNOPSIS

```
#include <string.h>

size_t strcspn(const char *s1, const char *s2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *strcspn*( ) function shall compute the length (in bytes) of the maximum initial segment of the string pointed to by *s1* which consists entirely of bytes *not* from the string pointed to by *s2*.

CX The *strcspn*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

The *strcspn*( ) function shall return the length of the computed segment of the string pointed to by *s1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strspn( )

XBD **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The RETURN VALUE section is updated to indicate that *strcspn*( ) returns the length of *s1*, and not *s1* itself as was previously stated.

Issue 6

The Open Group Corrigendum U030/1 is applied. The text of the RETURN VALUE section is updated to indicate that the computed segment length is returned, not the *s1* length.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *strcspn*( ) does not change the setting of *errno* on valid input.

69319 **NAME**

69320 strdup, strndup — duplicate a specific number of bytes from a string

69321 **SYNOPSIS**

69322 CX
```
#include <string.h>

char *strdup(const char *s);
char *strndup(const char *s, size_t size);
```

69325 **DESCRIPTION**

69326 The *strdup*( ) function shall return a pointer to a new string, which is a duplicate of the string
69327 pointed to by *s*. The returned pointer can be passed to *free*( ). A null pointer is returned if the
69328 new string cannot be created.

69329 The *strndup*( ) function shall be equivalent to the *strdup*( ) function, duplicating the provided *s* in
69330 a new block of memory allocated as if by using *malloc*( ), with the exception being that *strndup*( )
69331 copies at most *size* bytes from the array *s* into the newly allocated memory, terminating the new
69332 string with a null byte. If *s* contains a null terminator within the first *size* bytes, all bytes in *s* up
69333 to and including the null terminator shall be copied into the new memory buffer. The *strndup*( )
69334 function shall not examine more than *size* bytes of the array pointed to by *s*. The newly created
69335 string shall always be properly terminated.

69336 **RETURN VALUE**

69337 The *strdup*( ) function shall return a pointer to a new string on success. Otherwise, it shall return
69338 a null pointer and set *errno* to indicate the error.

69339 Upon successful completion, the *strndup*( ) function shall return a pointer to the newly allocated
69340 memory containing the duplicated string. Otherwise, it shall return a null pointer and set *errno*
69341 to indicate the error.

69342 **ERRORS**

69343 These functions shall fail if:

69344 [ENOMEM] Storage space available is insufficient.

69345 **EXAMPLES**

69346 None.

69347 **APPLICATION USAGE**

69348 For functions that allocate memory as if by *malloc*( ), the application should release such memory
69349 when it is no longer required by a call to *free*( ). For *strdup*( ) and *strndup*( ), this is the return
69350 value.

69351 Implementations are free to *malloc*( ) a buffer containing either (*size* + 1) bytes or (*strnlen*(*s*, *size*)
69352 + 1) bytes. Applications should not assume that *strndup*( ) will allocate (*size* + 1) bytes when
69353 *strlen*(*s*) is smaller than *size*.

69354 **RATIONALE**

69355 None.

69356 **FUTURE DIRECTIONS**

69357 None.

69358 **SEE ALSO**

69359 *free*( ), *wcsdup*( )

69360 XBD **<string.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 7

Austin Group Interpretation 1003.1-2001 #044 is applied, changing the ``may fail'' [ENOMEM] error to become a ``shall fail'' error.

The *strdup*( ) function is moved from the XSI option to the Base.

The *strndup*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The APPLICATION USAGE section is updated to clarify that memory is allocated as if by *malloc*( ).

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0338 [738] is applied.

Issue 8

Austin Group Defect 1019 is applied, clarifying that the *strndup*( ) argument *s* need not point to a null-terminated string.

NAME

strerror, strerror_l, strerror_r — get error message string

SYNOPSIS

```
#include <string.h>

char *strerror(int errnum);
CX char *strerror_l(int errnum, locale_t locale);
int strerror_r(int errnum, char *strerrbuf, size_t buflen);
```

DESCRIPTION

CX For *strerror*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *strerror*( ) function shall map the error number in *errnum* to a locale-dependent error message string and shall return a pointer to it. Typically, the values for *errnum* come from *errno*, but *strerror*( ) shall map any value of type **int** to a message.

CX The application shall not modify the string returned. The returned string pointer might be CX invalidated or the string content might be overwritten by a subsequent call to *strerror*( ), or by a subsequent call to *strerror_l*( ) in the same thread. The returned pointer and the string content might also be invalidated if the calling thread is terminated.

CX The string may be overwritten by a subsequent call to *strerror_l*( ) in the same thread.

The contents of the error message strings returned by *strerror*( ) should be determined by the setting of the *LC_MESSAGES* category in the current locale.

The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls *strerror*( ).

CX The *strerror*( ) and *strerror_l*( ) functions shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error of *strerror*( ), an application wishing to check for error situations should set *errno* to 0, then call *strerror*( ), then check *errno*. Similarly, since *strerror_l*( ) is required to return a string for some errors, an application wishing to check for all error situations should set *errno* to 0, then call *strerror_l*( ), then check *errno*.

The *strerror*( ) function need not be thread-safe; however, *strerror*( ) shall avoid data races with all other functions.

CX The *strerror_l*( ) function shall map the error number in *errnum* to a locale-dependent error message string in the locale represented by *locale* and shall return a pointer to it.

The *strerror_r*( ) function shall map the error number in *errnum* to a locale-dependent error message string and shall return the string in the buffer pointed to by *strerrbuf*, with length *buflen*.

CX If the value of *errnum* is a valid error number, the message string shall indicate what error occurred; if the value of *errnum* is zero, the message string shall either be an empty string or indicate that no error occurred; otherwise, if these functions complete successfully, the message string shall indicate that an unknown error occurred.

CX The behavior is undefined if the *locale* argument to *strerror_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

Upon completion, whether successful or not, *strerror*( ) shall return a pointer to the generated message string. CX On error *errno* may be set, but no return value is reserved to indicate an error.

Upon successful completion, *strerror_l*( ) shall return a pointer to the generated message string. If *errnum* is not a valid error number, *errno* may be set to [EINVAL], but a pointer to a message string shall still be returned. If any other error occurs, *errno* shall be set to indicate the error and a null pointer shall be returned.

Upon successful completion, *strerror_r*( ) shall return 0. Otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

CX [EINVAL] The value of *errnum* is neither a valid error number nor zero.

The *strerror_r*( ) function shall fail if:

CX [ERANGE] Insufficient storage was supplied via *strerrbuf* and *buflen* to contain the generated message string.

EXAMPLES

None.

APPLICATION USAGE

Historically in some implementations, calls to *perror*( ) would overwrite the string that the pointer returned by *strerror*( ) points to. Such implementations did not conform to the ISO C standard; however, application developers should be aware of this behavior if they wish their applications to be portable to such implementations.

Applications should use *strerror_l*( ) rather than *strerror*( ) or *strerror_r*( ) to avoid thread safety and possible alternative (non-conforming) versions of these functions in some implementations.

RATIONALE

The *strerror_l*( ) function is required to be thread-safe, thereby eliminating the need for an equivalent to the *strerror_r*( ) function.

Earlier versions of this standard did not explicitly require that the error message strings returned by *strerror*( ) and *strerror_r*( ) provide any information about the error. This version of the standard requires a meaningful message for any successful completion.

Since no return value is reserved to indicate a *strerror*( ) error, but all calls (whether successful or not) must return a pointer to a message string, on error *strerror*( ) can return a pointer to an empty string or a pointer to a meaningful string that can be printed.

Note that the [EINVAL] error condition is a may fail error. If an invalid error number is supplied as the value of *errnum*, applications should be prepared to handle any of the following:

1. Error (with no meaningful message): *errno* is set to [EINVAL], the return value is a pointer to an empty string.
2. Successful completion: *errno* is unchanged and the return value points to a string like `"unknown error"` or `"error number xxx"` (where *xxx* is the value of *errnum*).
3. Combination of #1 and #2: *errno* is set to [EINVAL] and the return value points to a string like `"unknown error"` or `"error number xxx"` (where *xxx* is the value of *errnum*). Since applications frequently use the return value of *strerror*( ) as an argument to functions like *fprintf*( ) (without checking the return value) and since applications have no way to parse an error message string to determine whether *errnum* represents a valid

69462 error number, implementations are encouraged to implement #3. Similarly,
69463 implementations are encouraged to have *strerror_r*( ) return [EINVAL] and put a string
69464 like `"unknown error"` or `"error number xxx"` in the buffer pointed to by *strerrbuf*
69465 when the value of *errnum* is not a valid error number.

69466 Additionally, implementations are encouraged to null terminate *strerrbuf* when failing with
69467 [ERANGE] for any size other than *buflen* of zero.

69468 Some applications rely on being able to set *errno* to 0 before calling a function with no reserved
69469 value to indicate an error, then call *strerror*(*errno*) afterwards to detect whether an error occurred
69470 (because *errno* changed) or to indicate success (because *errno* remained zero). This usage pattern
69471 requires that *strerror*(0) succeed with useful results. Previous versions of the standard did not
69472 specify the behavior when *errnum* is zero.

69473 **FUTURE DIRECTIONS**

69474 None.

69475 **SEE ALSO**

69476 *perror*( )

69477 XBD **<string.h>**

69478 **CHANGE HISTORY**

69479 First released in Issue 3.

69480 **Issue 5**

69481 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

69482 A note indicating that the *strerror*( ) function need not be reentrant is added to the
69483 DESCRIPTION.

69484 **Issue 6**

69485 Extensions beyond the ISO C standard are marked.

69486 The following new requirements on POSIX implementations derive from alignment with the
69487 Single UNIX Specification:

69488 • In the RETURN VALUE section, the fact that *errno* may be set is added.

69489 • The [EINVAL] optional error condition is added.

69490 The normative text is updated to avoid use of the term ``must'' for application requirements.

69491 The *strerror_r*( ) function is added in response to IEEE PASC Interpretation 1003.1c #39.

69492 The *strerror_r*( ) function is marked as part of the Thread-Safe Functions option.

69493 **Issue 7**

69494 Austin Group Interpretation 1003.1-2001 #072 is applied, updating the ERRORS section.

69495 Austin Group Interpretation 1003.1-2001 #156 is applied.

69496 Austin Group Interpretation 1003.1-2001 #187 is applied, clarifying the behavior when the
69497 generated error message is an empty string.

69498 SD5-XSH-ERN-191 is applied, updating the APPLICATION USAGE section.

69499 The *strerror_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API
69500 Set Part 4.

69501 The *strerror_r*( ) function is moved from the Thread-Safe Functions option to the Base.

69502 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0595 [75], XSH/TC1-2008/0596 [447],

XSH/TC1-2008/0597 [382,428], XSH/TC1-2008/0598 [283], XSH/TC1-2008/0599 [382,428], XSH/TC1-2008/0600 [283], and XSH/TC1-2008/0601 [382,428] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0339 [656] is applied.

Issue 8

Austin Group Defect 398 is applied, changing the [ERANGE] error from ``may fail'' to ``shall fail''.

Austin Group Defect 655 is applied, changing the APPLICATION USAGE section.

Austin Group Defect 1302 is applied, aligning the *strerror*( ) function with the ISO/IEC 9899:2018 standard.

NAME

strfmon, strfmon_l — convert monetary value to a string

SYNOPSIS

```
#include <monetary.h>

ssize_t strfmon(char *restrict s, size_t maxsize,
    const char *restrict format, ...);
ssize_t strfmon_l(char *restrict s, size_t maxsize,
    locale_t locale, const char *restrict format, ...);
```

DESCRIPTION

The *strfmon*( ) function shall place characters into the array pointed to by *s* as controlled by the string pointed to by *format*. No more than *maxsize* bytes are placed into the array.

The format is a character string, beginning and ending in its initial state, if any, that contains two types of objects: *plain characters*, which are simply copied to the output stream, and *conversion specifications*, each of which shall result in the fetching of zero or more arguments which are converted and formatted. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments are simply ignored.

The application shall ensure that a conversion specification consists of the following sequence:

- A '%' character
- Optional flags
- Optional field width
- Optional left precision
- Optional right precision
- A required conversion specifier character that determines the conversion to be performed

The *strfmon_l*( ) function shall be equivalent to the *strfmon*( ) function, except that the locale data used is from the locale represented by *locale*.

Flags

One or more of the following optional flags can be specified to control the conversion:

=f An '=' followed by a single character *f* which is used as the numeric fill character. In order to work with precision or width counts, the fill character shall be a single byte character; if not, the behavior is undefined. The default numeric fill character is the <space>. This flag does not affect field width filling which always uses the <space>. This flag is ignored unless a left precision (see below) is specified.

^ Do not format the currency amount with grouping characters. The default is to insert the grouping characters if defined for the current locale.

+ or (Specify the style of representing positive and negative currency amounts. Only one of '+' or '(' may be specified.

If '+' is specified, the locale's **positive_sign** and **negative_sign** values shall be used (for example, in many locales, the empty string if positive and '-' if negative). However, if both values would be returned by *localeconv*( ) as empty strings, *strfmon*( ) shall fail. The placement of the signs (if not empty) shall depend on the locale settings:

- For the `n` conversion specifier, the placement specified by the locale's **p_sign_posn** and **n_sign_posn** values shall be used.
- For the `i` conversion specifier, the placement specified by the locale's **int_p_sign_posn** and **int_n_sign_posn** values shall be used.

If a sign's placement cannot be determined from these locale values because a value that needs to be used would be returned by *localeconv*( ) as 0 or {CHAR_MAX}, the sign shall be placed as if the relevant value was 1.

If `'('` is specified, negative amounts shall be enclosed within parentheses and the locale's **positive_sign** and **negative_sign** values shall not be used.

If neither flag is specified, the style used shall depend on the locale settings:

- For the `n` conversion specifier, the style specified by the locale's **p_sign_posn** and **n_sign_posn** values shall be used.
- For the `i` conversion specifier, the style specified by the locale's **int_p_sign_posn** and **int_n_sign_posn** values shall be used.

If the style cannot be determined from these locale values because a value that needs to be used would be returned by *localeconv*( ) as {CHAR_MAX}, the style used shall be that specified for the `'+'` flag; if this would cause *strfmon*( ) to fail because the locale's **positive_sign** and **negative_sign** values would both be returned by *localeconv*( ) as empty strings, *strfmon*( ) shall behave as if the **negative_sign** value was the string `"-"`.

`!` Suppress the currency symbol from the output conversion.

– Specify the alignment. If this flag is present the result of the conversion is left-justified (padded to the right) rather than right-justified. This flag shall be ignored unless a field width (see below) is specified.

Field Width

w A decimal digit string *w* specifying a minimum field width in bytes in which the result of the conversion is right-justified (or left-justified if the flag `'-'` is specified). The default is 0.

Left Precision

#*n* A `'#'` followed by a decimal digit string *n* specifying a maximum number of digits expected to be formatted to the left of the radix character. This option can be used to keep the formatted output from multiple calls to the *strfmon*( ) function aligned in the same columns. It can also be used to fill unused positions with a special character as in `"$***123.45"`. This option causes an amount to be formatted as if it has the number of digits specified by *n*. If more than *n* digit positions are required, this conversion specification is ignored. Digit positions in excess of those actually required are filled with the numeric fill character (see the *=f* flag above).

If grouping has not been suppressed with the `'^'` flag, and it is defined for the current locale, grouping separators are inserted before the fill characters (if any) are added. Grouping separators are not applied to fill characters even if the fill character is a digit.

To ensure alignment, any characters appearing before or after the number in the formatted output such as currency or sign symbols are padded as necessary with <space> characters to make their positive and negative formats an equal length.

69595 **Right Precision**

69596 `.p` A <period> followed by a decimal digit string *p* specifying the number of digits after
69597 the radix character. If the value of the right precision *p* is 0, no radix character appears.
69598 If a right precision is not included, a default specified by the current locale is used. The
69599 amount being formatted is rounded to the specified number of digits prior to
69600 formatting.

69601 **Conversion Specifier Characters**

69602 The conversion specifier characters and their meanings are:

69603 `i` The **double** argument is formatted according to the locale's international currency
69604 format (for example, in the US: USD 1,234.56). If the argument is ±Inf or NaN, the result
69605 of the conversion is unspecified.

69606 `n` The **double** argument is formatted according to the locale's national currency format
69607 (for example, in the US: $1,234.56). If the argument is ±Inf or NaN, the result of the
69608 conversion is unspecified.

69609 `%` Convert to a `'%'`; no argument is converted. The entire conversion specification shall
69610 be `%%`.

69611 **Locale Information**

69612 The *LC_MONETARY* category of the current locale affects the behavior of this function including
69613 the monetary radix character (which may be different from the numeric radix character affected
69614 by the *LC_NUMERIC* category), the grouping separator, the currency symbols, and formats. The
69615 international currency symbol should be conformant with the ISO 4217: 2015 standard.

69616 If the value of *maxsize* is greater than {SSIZE_MAX}, the result is implementation-defined.

69617 The behavior is undefined if the *locale* argument to *strfmon_l*( ) is the special locale object
69618 LC_GLOBAL_LOCALE or is not a valid locale object handle.

69619 **RETURN VALUE**

69620 If all conversions are successful and the total number of resulting bytes including the
69621 terminating null byte is not more than *maxsize*, these functions shall return the number of bytes
69622 placed into the array pointed to by *s*, not including the terminating NUL character. Otherwise,
69623 –1 shall be returned, the contents of the array are unspecified, and *errno* shall be set to indicate
69624 the error.

69625 **ERRORS**

69626 These functions shall fail if:

69627 [E2BIG] Conversion stopped due to lack of space in the buffer.

69628 [EINVAL] The `'+'` flag was included in a conversion specification and the locale's
69629 **positive_sign** and **negative_sign** values would both be returned by
69630 *localeconv*( ) as empty strings.

EXAMPLES

Given a locale for the US and the values 123.45, –123.45, and 3456.781, the following output might be produced. Square brackets ("`[ ]`") are used in this example to delimit the output.

`%n`	`[$123.45]`
	`[-$123.45]`
	`[$3,456.78]`
		Default formatting

`%11n`	`[    $123.45]`
	`[   -$123.45]`
	`[  $3,456.78]`
		Right align within an 11-character field

`%#5n`	`[ $   123.45]`
	`[-$   123.45]`
	`[ $ 3,456.78]`
		Aligned columns for values up to 99 999

`%=*#5n`	`[ $***123.45]`
	`[-$***123.45]`
	`[ $*3,456.78]`
		Specify a fill character

`%=0#5n`	`[ $000123.45]`
	`[-$000123.45]`
	`[ $03,456.78]`
		Fill characters do not use grouping even if the fill character is a digit

`%^#5n`	`[ $  123.45]`
	`[-$  123.45]`
	`[ $ 3456.78]`
		Disable the grouping separator

`%^#5.0n`	`[ $  123]`
	`[-$  123]`
	`[ $ 3457]`
		Round off to whole units

`%^#5.4n`	`[ $  123.4500]`
	`[-$  123.4500]`
	`[ $ 3456.7810]`
		Increase the precision

`%(#5n`	`[ $   123.45 ]`
	`[($   123.45)]`
	`[ $ 3,456.78 ]`
		Use an alternative pos/neg style

`%!(#5n`	`[    123.45 ]`
	`[(   123.45)]`
	`[  3,456.78 ]`
		Disable the currency symbol

`%-14#5.4n`	`[ $   123.4500 ]`
	`[-$   123.4500 ]`
	`[ $ 3,456.7810 ]`
		Left-justify the output

`%14#5.4n`	`[  $   123.4500]`
	`[ -$   123.4500]`
	`[  $ 3,456.7810]`
		Corresponding right-justified output

See also the EXAMPLES section in *fprintf*( ).

APPLICATION USAGE

The `'+'` flag should be used with care, because if the locale's **positive_sign** and **negative_sign** values are both empty strings, there is no way to distinguish negative from positive values with signs and therefore *strfmon*( ) fails. If the application has a preference for signs but parentheses are acceptable, it should try *strfmon*( ) with the `'+'` flag first, and if it fails with [EINVAL] then

repeat the call without the '+' flag.

RATIONALE

The [EINVAL] error condition applies only when the '+' flag is used because this flag indicates that the application requires the use of signs, and if there are no signs in the locale data then this requirement cannot be satisfied. When neither '+' nor '(' is used, the application is requesting whatever formatting is appropriate for the locale, and so *strfmon*( ) has a fallback of using a '−' sign for negative values in cases where the locale data does not indicate parentheses should be used and has no signs.

FUTURE DIRECTIONS

Lowercase conversion characters are reserved for future standards use and uppercase for implementation-defined use.

SEE ALSO

fprintf( ), *localeconv*( )

XBD **<monetary.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

Moved from ENHANCED I18N to BASE.

The [ENOSYS] error is removed.

Text is added to the DESCRIPTION warning about values of *maxsize* that are greater than {SSIZE_MAX}.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

The **restrict** keyword is added to the *strfmon*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

The EXAMPLES section is reworked, clarifying the output format.

Issue 7

SD5-XSH-ERN-29 is applied, updating the examples for %(#5n and %!(#5n.

SD5-XSH-ERN-233 is applied, changing the definition of the '+' or '(' flags to refer to multiple locales.

The *strfmon*( ) function is moved from the XSI option to the Base.

The *strfmon_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0602 [302], XSH/TC1-2008/0603 [283], and XSH/TC1-2008/0604 [283] are applied.

Issue 8

Austin Group Defect 1199 is applied, changing the requirements for the '+' and '(' flags and adding the [EINVAL] error.

NAME

strftime, strftime_l — convert date and time to a string

SYNOPSIS

```
#include <time.h>

size_t strftime(char *restrict s, size_t maxsize,
    const char *restrict format, const struct tm *restrict timeptr);
CX size_t strftime_l(char *restrict s, size_t maxsize,
    const char *restrict format, const struct tm *restrict timeptr,
    locale_t locale);
```

DESCRIPTION

CX For *strftime*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *strftime*( ) function shall place bytes into the array pointed to by *s* as controlled by the string pointed to by *format*. The application shall ensure that the format is a character string, beginning and ending in its initial shift state, if any. The format string consists of zero or more conversion specifications and ordinary characters.

Each conversion specification is introduced by the '%' character after which the following appear in sequence:

CX
- An optional flag:

  0 The zero character ('0'), which specifies that the character used as the padding character is '0',

  + The <plus-sign> character ('+'), which specifies that the character used as the padding character is '0', and that if and only if the field being produced consumes more than four bytes to represent a year (for %F, %G, or %Y) or more than two bytes to represent the year divided by 100 (for %C) then a leading <plus-sign> character shall be included if the year being processed is greater than or equal to zero or a leading <hyphen-minus> character ('−') shall be included if the year is less than zero.

  The default padding character is unspecified.

- An optional minimum field width. If the converted value, including any leading '+' or '−' sign, has fewer bytes than the minimum field width and the padding character is not the NUL character, the output shall be padded on the left (after any leading '+' or '−' sign) with the padding character.
- An optional E or O modifier.
- A terminating conversion specifier character that indicates the type of conversion to be applied.

CX The results are unspecified if more than one flag character is specified, a flag character is specified without a minimum field width; a minimum field width is specified without a flag character; a modifier is specified with a flag or with a minimum field width; or if a minimum field width is specified for any conversion specifier other than C, F, G, or Y.

All ordinary characters (including the terminating NUL character) are copied unchanged into the array. If copying takes place between objects that overlap, the behavior is undefined. No more than *maxsize* bytes are placed into the array. Each conversion specifier is replaced by appropriate characters as described in the following list. The appropriate characters are

determined using the *LC_TIME* category of the current locale and by the values of zero or more members of the broken-down time structure pointed to by *timeptr*, as specified in brackets in the description. If any of the specified values are outside the normal range, the characters stored are unspecified.

CX The *strftime_l*( ) function shall be equivalent to the *strftime*( ) function, except that the locale data used is from the locale represented by *locale*.

Local timezone information shall be set as though *strftime*( ) called *tzset*( ).

The following conversion specifiers shall be supported:

`a` Replaced by the locale's abbreviated weekday name. [*tm_wday*]

`A` Replaced by the locale's full weekday name. [*tm_wday*]

`b` Replaced by the locale's abbreviated month name. [*tm_mon*]

`B` Replaced by the locale's full month name. [*tm_mon*]

`c` Replaced by the locale's appropriate date and time representation. (See the Base Definitions volume of POSIX.1-2024, **<time.h>**.)

`C` Replaced by the year divided by 100 and truncated to an integer, as a decimal number. [*tm_year*]

If a minimum field width is not specified:

- If the year is between 0 and 9999 inclusive, two characters shall be placed into the array pointed to by *s*, including a leading `'0'` if there would otherwise be only a single digit.
- CX If the year is less than 0 or greater than 9999, the number of characters placed into the array pointed to by *s* shall be the number of digits and leading sign characters (if any) in the result of dividing the year by 100 and truncating, or two, whichever is greater.

CX If a minimum field width is specified, the number of characters placed into the array pointed to by *s* shall be the number of digits and leading sign characters (if any) in the result of dividing the year by 100 and truncating, or the minimum field width, whichever is greater.

`d` Replaced by the day of the month as a decimal number [01,31]. [*tm_mday*]

`D` Equivalent to `%m/%d/%y`. [*tm_mon*, *tm_mday*, *tm_year*]

`e` Replaced by the day of the month as a decimal number [1,31]; a single digit is preceded by a space. [*tm_mday*]

`F` Equivalent to %Y-%m-%d if no flag and no minimum field width are specified. (For years between 1000 and 9999 inclusive this provides the ISO 8601-1:2019 standard complete representation, extended format date representation of a specific day.) [*tm_year*, *tm_mon*, *tm_mday*]

CX If a minimum field width of *x* is specified, the year shall be output as if by the `Y` specifier (described below) with whatever flag was given and a minimum field width of *x*−6. If *x* is less than 6, the behavior shall be as if *x* equalled 6.

If the minimum field width is specified to be 10, and the year is four digits long, then the output string produced shall match the ISO 8601-1:2019 standard subclause 4.1.2.2 complete representation, extended format date representation of a specific day. If a +

flag is specified, a minimum field width of *x* is specified, and *x*–7 bytes are sufficient to hold the digits of the year (not including any needed sign character), then the output shall match the ISO 8601-1:2019 standard subclause 4.1.2.4 complete representation, expanded format date representation of a specific day.

`g` Replaced by the last 2 digits of the week-based year (see below) as a decimal number [00,99]. [*tm_year*, *tm_wday*, *tm_yday*]

`G` Replaced by the week-based year (see below) as a decimal number (for example, 1977). [*tm_year*, *tm_wday*, *tm_yday*]

CX If a minimum field width is specified, the number of characters placed into the array pointed to by *s* shall be the number of digits and leading sign characters (if any) in the year, or the minimum field width, whichever is greater.

`h` Equivalent to `%b`. [*tm_mon*]

`H` Replaced by the hour (24-hour clock) as a decimal number [00,23]. [*tm_hour*]

`I` Replaced by the hour (12-hour clock) as a decimal number [01,12]. [*tm_hour*]

`j` Replaced by the day of the year as a decimal number [001,366]. [*tm_yday*]

`m` Replaced by the month as a decimal number [01,12]. [*tm_mon*]

`M` Replaced by the minute as a decimal number [00,59]. [*tm_min*]

`n` Replaced by a <newline>.

`p` Replaced by the locale's equivalent of either a.m. or p.m. [*tm_hour*]

CX `r` Replaced by the time in 12-hour clock notation; if the 12-hour format is not supported in the locale, this shall be either an empty string or the time in a 24-hour clock notation. In the POSIX locale this shall be equivalent to `%I:%M:%S %p`. [*tm_hour*, *tm_min*, *tm_sec*]

`R` Replaced by the time in 24-hour notation (`%H:%M`). [*tm_hour*, *tm_min*]

CX `s` Replaced by the number of seconds since the Epoch as a decimal number, calculated as described for *mktime*( ). [*tm_year*, *tm_mon*, *tm_mday*, *tm_hour*, *tm_min*, *tm_sec*, *tm_isdst*]

`S` Replaced by the second as a decimal number [00,60]. [*tm_sec*]

`t` Replaced by a <tab>.

`T` Replaced by the time (`%H:%M:%S`). [*tm_hour*, *tm_min*, *tm_sec*]

`u` Replaced by the weekday as a decimal number [1,7], with 1 representing Monday. [*tm_wday*]

`U` Replaced by the week number of the year as a decimal number [00,53]. The first Sunday of January is the first day of week 1; days in the new year before this are in week 0. [*tm_year*, *tm_wday*, *tm_yday*]

`V` Replaced by the week number of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1. Both January 4th and the first Thursday of January are always in week 1. [*tm_year*, *tm_wday*, *tm_yday*]

`w` Replaced by the weekday as a decimal number [0,6], with 0 representing Sunday. [*tm_wday*]

`W` Replaced by the week number of the year as a decimal number [00,53]. The first Monday of January is the first day of week 1; days in the new year before this are in week 0. [*tm_year*, *tm_wday*, *tm_yday*]

`x` Replaced by the locale's appropriate date representation. (See the Base Definitions volume of POSIX.1-2024, **<time.h>**.)

`X` Replaced by the locale's appropriate time representation. (See the Base Definitions volume of POSIX.1-2024, **<time.h>**.)

`y` Replaced by the last two digits of the year as a decimal number [00,99]. [*tm_year*]

`Y` Replaced by the year as a decimal number (for example, 1997). [*tm_year*]

CX If a minimum field width is specified, the number of characters placed into the array pointed to by *s* shall be the number of digits and leading sign characters (if any) in the year, or the minimum field width, whichever is greater.

`z` Replaced by the offset from UTC in the ISO 8601-1:2019 standard format (`+hhmm` or `−hhmm`), or by no characters if no timezone is determinable. For example, `"−0430"` means 4 hours 30 minutes behind UTC (west of Greenwich). If *tm_isdst* is zero, the standard time offset is used. If *tm_isdst* is greater than zero, the daylight saving time offset is used. If *tm_isdst* is negative, no characters are returned. [*tm_isdst*, *tm_gmtoff*]

`Z` Replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists. [*tm_isdst*, *tm_zone*]

`%` Replaced by `%`.

If a conversion specification does not correspond to any of the above, the behavior is undefined.

CX If a **struct tm** broken-down time structure is created by *localtime*( ) or *localtime_r*( ), or modified by *mktime*( ), and the value of *TZ* is subsequently modified, the results of the `%Z` and `%z` *strftime*( ) conversion specifiers are undefined, when *strftime*( ) is called with such a broken-down time structure.

If a **struct tm** broken-down time structure is created or modified by *gmtime*( ) or *gmtime_r*( ), it is unspecified whether the result of the `%Z` and `%z` conversion specifiers shall refer to UTC or the current local timezone, when *strftime*( ) is called with such a broken-down time structure.

Modified Conversion Specifiers

Some conversion specifiers can be modified by the `E` or `O` modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist for the current locale (see ERA in XBD Section 7.3.5, on page 152), the behavior shall be as if the unmodified conversion specification were used.

`%Ec` Replaced by the locale's alternative appropriate date and time representation.

`%EC` Replaced by the name of the base year (period) in the locale's alternative representation.

`%Ex` Replaced by the locale's alternative date representation.

`%EX` Replaced by the locale's alternative time representation.

`%Ey` Replaced by the offset from `%EC` (year only) in the locale's alternative representation.

`%EY` Replaced by the full alternative year representation.

CX `%Ob` Replaced by the locale's abbreviated alternative month name.

CX `%OB` Replaced by the locale's alternative appropriate full month name.

`%Od` Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading zeros if there is any alternative symbol for zero; otherwise, with leading <space> characters.

`%Oe` Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading <space> characters.

`%OH` Replaced by the hour (24-hour clock) using the locale's alternative numeric symbols.

`%OI` Replaced by the hour (12-hour clock) using the locale's alternative numeric symbols.

`%Om` Replaced by the month using the locale's alternative numeric symbols.

`%OM` Replaced by the minutes using the locale's alternative numeric symbols.

`%OS` Replaced by the seconds using the locale's alternative numeric symbols.

`%Ou` Replaced by the weekday as a number in the locale's alternative representation (Monday=1).

`%OU` Replaced by the week number of the year (Sunday as the first day of the week, rules corresponding to `%U`) using the locale's alternative numeric symbols.

`%OV` Replaced by the week number of the year (Monday as the first day of the week, rules corresponding to `%V`) using the locale's alternative numeric symbols.

`%Ow` Replaced by the number of the weekday (Sunday=0) using the locale's alternative numeric symbols.

`%OW` Replaced by the week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.

`%Oy` Replaced by the year (offset from `%C`) using the locale's alternative numeric symbols.

`%g`, `%G`, and `%V` give values according to the ISO 8601-1:2019 standard week-based year. In this system, weeks begin on a Monday and week 1 of the year is the week that includes January 4th, which is also the week that includes the first Thursday of the year, and is also the first week that contains at least four days in the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year; thus, for Saturday 2nd January 1999, `%G` is replaced by 1998 and `%V` is replaced by 53. If December 29th, 30th, or 31st is a Monday, it and any following days are part of week 1 of the following year. Thus, for Tuesday 30th December 1997, `%G` is replaced by 1998 and `%V` is replaced by 01.

If a conversion specifier is not one of the above, the behavior is undefined.

CX The behavior is undefined if the *locale* argument to *strftime_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

If successful, these functions shall return the number of bytes placed into the array pointed to by *s*, not including the terminating NUL character. CX If successful, *errno* shall not be changed. CX Otherwise, 0 shall be returned, *errno* shall be set to indicate the error, and the contents of the array are unspecified.

ERRORS

CX These functions shall fail if:

[ERANGE] The total number of resulting bytes including the terminating NUL character is more than *maxsize*.

These functions may fail if:

[EINVAL] The *format* string includes a `%s` conversion and the number of seconds since the Epoch would be negative.

[EOVERFLOW] The *format* string includes a `%s` conversion and the number of seconds since the Epoch cannot be represented in a **time_t**.

EXAMPLES

Getting a Localized Date String

The following example first sets the locale to the user's default. The locale information will be used in the *nl_langinfo*( ) and *strftime*( ) functions. The *nl_langinfo*( ) function returns the localized date string which specifies how the date is laid out. The *strftime*( ) function takes this information and, using the **tm** structure for values, places the date and time information into *datestring*.

```
#include <time.h>
#include <locale.h>
#include <langinfo.h>
...
struct tm *tm;
char datestring[256];
...
setlocale (LC_ALL, "");
...
strftime (datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
...
```

APPLICATION USAGE

A return value of 0 may indicate either success or failure if a format is the empty string or consists of conversion specifications such as `%p` or `%Z` that are replaced by no characters in the current locale or because of the current setting of *tzname*[], respectively. To distinguish between success and failure when *strftime()*( ) returns 0, an application can set *errno* to 0 before calling *strftime*( ) and test whether *errno* is 0 afterwards.

The range of values for `%S` is [00,60] rather than [00,59] to allow for the occasional leap second.

Some of the conversion specifications are duplicates of others. They are included for compatibility with *nl_cxtime*( ) and *nl_ascxtime*( ), which were published in Issue 2.

The `%C`, `%F`, `%G`, and `%Y` format specifiers in *strftime*( ) always print full values, but the *strptime*( ) `%C`, `%F`, and `%Y` format specifiers only scan two digits (assumed to be the first two digits of a four-digit year) for `%C` and four digits (assumed to be the entire (four-digit) year) for `%F` and `%Y`. This mimics the behavior of *printf*( ) and *scanf*( ); that is:

```
printf("%2d", x = 1000);
```

prints `"1000"`, but:

```
scanf(%2d", &x);
```


69964 when given "1000" as input will only store 10 in *x*). Applications using extended ranges of
69965 years must be sure that the number of digits specified for scanning years with *strptime*( ) matches
69966 the number of digits that will actually be present in the input stream. Historic implementations
69967 of the %Y conversion specification (with no flags and no minimum field width) produced
69968 different output formats. Some always produced at least four digits (with 0 fill for years from 0
69969 through 999) while others only produced the number of digits present in the year (with no fill
69970 and no padding). These two forms can be produced with the '0' flag and a minimum field
69971 width options using the conversions specifications %04Y and %01Y, respectively. Similarly,
69972 because %Y is part of %F, field widths of 10 and 7 (%010F, %07F), respectively, produce the same
69973 effect in the year portion of the %F conversion result.

69974 In the past, the C and POSIX standards specified that %F produced an ISO 8601-1:2019 standard
69975 date format, but didn't specify which one. For years in the range [1000,9999], POSIX.1-2024
69976 requires that the output produced match the ISO 8601-1:2019 standard complete representation
69977 extended format (YYYY-MM-DD) and for years greater than 9999 produce output that matches
69978 the ISO 8601-1:2019 standard expanded representation extended format ±YYYYY-MM-DD). For
69979 years less than 1000, %F is not required to produce an ISO 8601-1:2019 standard format when
69980 used without specifying at least a minimum field width. As stated above, some implementations
69981 pad %Y conversions with zeros to four digits, in which case %F produces an ISO 8601-1:2019
69982 standard format; other implementations do not pad %Y with zeros, in which case %F does not
69983 produce an ISO 8601-1:2019 standard format. To fully meet ISO 8601-1:2019 standard
69984 requirements, the producer and consumer must agree on a date format that has a specific
69985 number of bytes reserved to hold the characters used to represent the years that is sufficiently
69986 large to hold all values that will be shared. For example, the %+13F conversion specification will
69987 produce output matching the format "±YYYYYY−MM−DD" (a leading '+' or '−' sign; a six-digit,
69988 0-filled year; a '−'; a two-digit, leading 0-filled month; another '−'; and the two-digit, leading
69989 0-filled day within the month).

69990 Note that if the year being printed is greater than 9999, the resulting string from the unadorned
69991 %F conversion specifications will not conform to the ISO 8601-1:2019 standard extended format,
69992 complete representation for a date and will instead be an extended format, expanded
69993 representation (presumably without the required agreement between the date's producer and
69994 consumer).

69995 In the C or POSIX locale, the E and O modifiers are ignored and the replacement strings for the
69996 following specifiers are:

69997 %a The first three characters of %A.

69998 %A One of Sunday, Monday, ..., Saturday.

69999 %b The first three characters of %B.

70000 %B One of January, February, ..., December.

70001 %c Equivalent to %a %b %e %T %Y.

70002 %p One of AM or PM.

70003 %r Equivalent to %I:%M:%S %p.

70004 %x Equivalent to %m/%d/%y.

70005 %X Equivalent to %T.

70006 %Z Implementation-defined.

RATIONALE

The %Y conversion specification to *strftime*( ) was frequently assumed to be a four-digit year, but the ISO C standard does not specify that %Y is restricted to any subset of allowed values from the *tm_year* field. Similarly, the %C conversion specification was assumed to be a two-digit field and the first part of the output from the %F conversion specification was assumed to be a four-digit field. Since *tm_year* is a signed **int** with a width of at least 32 bits and **time_t** is required to have a width of at least 64 bits (in conforming programming environments), these assumptions no longer hold.

POSIX.1-2024 now allows the format specifications %0xC, %0xF, %0xG, and %0xY (where 'x' is a string of decimal digits used to specify printing and scanning of a string of *x* decimal digits) with leading zero fill characters. Allowing applications to set the field width enables them to agree on the number of digits to be printed and scanned in the ISO 8601-1:2019 standard expanded representation of a year (for %F, %G, and %Y) or all but the last two digits of the year (for %C). This is based on a feature in some versions of GNU **libc**'s *strftime*( ). The GNU version allows specifying space, zero, or no-fill characters in *strftime*( ) format strings, but does not allow any flags to be specified in *strptime*( ) format strings. These implementations also allow these flags to be specified for any numeric field. POSIX.1-2024 only requires the zero fill flag ('0') and only requires that it be recognized when processing %C, %F, %G, and %Y specifications when a minimum field width is also specified. The '0' flag is the only flag needed to produce and scan the ISO 8601-1:2019 standard year fields using the extended format forms. POSIX.1-2024 also allows applications to specify the same flag and field width specifiers to be used in both *strftime*( ) and *strptime*( ) format strings for symmetry. Systems may provide other flag characters and may accept flags in conjunction with conversion specifiers other than %C, %F, %G, and %Y; but portable applications cannot depend on such extensions.

POSIX.1-2024 now also allows the format specifications %+xC, %+xF, %+xG, and %+xY (where 'x' is a string of decimal digits used to specify printing and scanning of a string of 'x' decimal digits) with leading zero fill characters and a leading '+' sign character if the year being converted is more than four digits or a minimum field width is specified that allows room for more than four digits for the year. This allows date providers and consumers to agree on a specific number of digits to represent a year as required by the ISO 8601-1:2019 standard expanded representation formats. The expanded representation formats all require the year to begin with a leading '+' or '−' sign. (All of these specifiers can also provide a leading '−' sign for negative years. Since negative years and the year 0 don't fit well with the Gregorian or Julian calendars, the normal ranges of dates start with year 1. The ISO C standard allows *tm_year* to assume values corresponding to years before year 1, but the use of such years provided unspecified results.)

Some earlier version of this standard specified that applications wanting to use *strptime*( ) to scan dates and times printed by *strftime*( ) should provide non-digit characters between fields to separate years from months and days. It also supported %F to print and scan the ISO 8601-1:2019 standard extended format, complete representation date for years 1 through 9999 (i.e., YYYY-MM-DD). However, many applications were written to print (using *strftime*( )) and scan (using *strptime*( )) dates written using the basic format complete representation (four-digit years) and truncated representation (two-digit years) specified by the ISO 8601-1:2019 standard representation of dates and times which do not have any separation characters between fields. The ISO 8601-1:2019 standard also specifies basic format expanded representation where the creator and consumer of these fields agree beforehand to represent years as leading zero-filled strings of an agreed length of more than four digits to represent a year (again with no separation characters when year, month, and day are all displayed). Applications producing and consuming expanded representations are encouraged to use the '+' flag and an appropriate maximum field width to scan the year including the leading sign.

Note that even without the '+' flag, years less than zero may be represented with a leading <hyphen-minus> for %F, %G, and %Y conversion specifications. Using negative years results in unspecified behavior.

If a format specification %+xF with the field width *x* greater than 11 is specified and the width is large enough to display the full year, the output string produced will match the ISO 8601-1:2019 standard subclause 4.1.2.4 expanded representation, extended format date representation for a specific day. (For years in the range [1,99 999], %+12F is sufficient for an agreed five-digit year with a leading sign using the ISO 8601-1:2019 standard expanded representation, extended format for a specific day "±YYYYY−MM−DD".) Note also that years less than 0 may produce a leading <hyphen-minus> character ('−') when using %Y or %C whether or not the '0' or '+' flags are used.

The difference between the '0' flag and the '+' flag is whether the leading '+' character will be provided for years >9999 as required for the ISO 8601-1:2019 standard extended representation format containing a year. For example:

Year	Conversion Specification	*strftime*() Output	*strptime*() Scan Back
1970	%Y	1970	1970
1970	%+4Y	1970	1970
27	%Y	27 or 0027	27
270	%Y	270 or 0270	270
270	%+4Y	0270	270
17	%C%y	0017	17
270	%C%y	0270	270
12345	%Y	12345	1234*
12345	%+4Y	+12345	123*
12345	%05Y	12345	12345
270	%+5Y or %+3C%y	+0270	270
12345	%+5Y or %+3C%y	+12345	1234*
12345	%06Y or %04C%y	012345	12345
12345	%+6Y or %+4C%y	+12345	12345
123456	%08Y or %06C%y	00123456	123456
123456	%+8Y or %+6C%y	+0123456	123456

In the cases above marked with a * in the *strptime*( ) scan back field, the implied or specified number of characters scanned by *strptime*( ) was less than the number of characters output by *strftime*( ) using the same format; so the remaining digits of the year were dropped when the output date produced by *strftime*( ) was scanned back in by *strptime*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

asctime( ), *clock*( ), *ctime*( ), *difftime*( ), *futimens*( ), *getdate*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strptime*( ), *time*( ), *tzset*( ), *uselocale*( )

XBD Section 7.3.5 (on page 152), **<time.h>**

70099 **CHANGE HISTORY**

70100 First released in Issue 3.

70101 **Issue 5**

70102 The description of `%OV` is changed to be consistent with `%V` and defines Monday as the first day
70103 of the week.

70104 The description of `%Oy` is clarified.

70105 **Issue 6**

70106 Extensions beyond the ISO C standard are marked.

70107 The Open Group Corrigendum U033/8 is applied. The `%V` conversion specifier is changed from
70108 ``Otherwise, it is week 53 of the previous year, and the next week is week 1'' to ``Otherwise, it is
70109 the last week of the previous year, and the next week is week 1''.

70110 The following new requirements on POSIX implementations derive from alignment with the
70111 Single UNIX Specification:

70112 - The `%C`, `%D`, `%e`, `%h`, `%n`, `%r`, `%R`, `%t`, and `%T` conversion specifiers are added.

70113 - The modified conversion specifiers are added for consistency with the ISO POSIX-2
70114 standard *date* utility.

70115 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

70116 - The *strftime*( ) prototype is updated.

70117 - The DESCRIPTION is extensively revised.

70118 - The `%z` conversion specifier is added.

70119 An example is added.

70120 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/60 is applied.

70121 **Issue 7**

70122 Austin Group Interpretation 1003.1-2001 #163 is applied.

70123 The *strftime_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API
70124 Set Part 4.

70125 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0605 [283], XSH/TC1-2008/0606 [283],
70126 XSH/TC1-2008/0607 [193], and XSH/TC1-2008/0608 [193] are applied.

70127 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0340 [584], XSH/TC2-2008/0341 [796],
70128 XSH/TC2-2008/0342 [584], and XSH/TC2-2008/0343 [584] are applied.

70129 **Issue 8**

70130 Austin Group Defects 169, 1386, and 1612 are applied, adding the `s` conversion and requiring
70131 *errno* to be unchanged on success and set on error.

70132 Austin Group Defects 258 and 1166 are applied, adding the `OB` and `Ob` modified conversions.

70133 Austin Group Defect 472 is applied, changing the description of the `C` conversion.

70134 Austin Group Defect 739 is applied, changing the `%F` conversion to match the ISO C standard
70135 when no flag and no minimum field width are specified.

70136 Austin Group Defect 1125 is applied, changing ``Local timezone information is used'' to ``Local
70137 timezone information shall be set''.

70138 Austin Group Defect 1253 is applied, changing ``daylight savings'' to ``daylight saving''.

Austin Group Defect 1307 is applied, changing the `r` conversion in relation to locales that do not support the 12-hour clock format.

Austin Group Defects 1313 and 1354 are applied, changing text relating to the ISO 8601-1: 2019 standard in the APPLICATION USAGE section.

Austin Group Defect 1462 is applied, changing the RATIONALE section.

Austin Group Defect 1533 is applied, adding *tm_gmtoff* and *tm_zone* to the **tm** structure.

Austin Group Defect 1562 is applied, clarifying that it is the application's responsibility to ensure that the format is a character string, beginning and ending in its initial shift state, if any.

NAME

strlcat, strlcpy — size-bounded string concatenation and copying

SYNOPSIS

CX
```
#include <string.h>

size_t strlcat(char *restrict dst, const char *restrict src,
    size_t dstsize);
size_t strlcpy(char *restrict dst, const char *restrict src,
    size_t dstsize);
```

DESCRIPTION

The *strlcpy*( ) and *strlcat*( ) functions copy and concatenate strings, stopping when either a NUL terminator in the source string is encountered or the specified full size of the destination buffer is reached. They NUL terminate the result if there is room. The application should ensure that room for the NUL terminator is included in *dstsize*.

The *strlcpy*( ) function shall copy not more than *dstsize* – 1 bytes from the string pointed to by *src* to the array pointed to by *dst*; a NUL byte in *src* and bytes that follow it shall not be copied. A terminating NUL byte shall be appended to the result, unless *dstsize* is 0. If copying takes place between objects that overlap, the behavior is undefined.

The *strlcat*( ) function shall append not more than *dstsize* – *strlen*(*dst*) – 1 bytes from the string pointed to by *src* to the end of the string pointed to by *dst*; a NUL byte in *src* and bytes that follow it shall not be appended. The initial byte of *src* shall overwrite the NUL byte at the end of *dst*. A terminating NUL byte shall be appended to the result, unless its location would be at or beyond *dst* + *dstsize*. If copying takes place between objects that overlap, the behavior is undefined.

The *strlcpy*( ) and *strlcat*( ) functions shall not change the setting of *errno* on valid input.

RETURN VALUE

Upon successful completion, the *strlcpy*( ) function shall return the length of the string pointed to by *src*; that is, the number of bytes in the string, not including the terminating NUL byte.

Upon successful completion, the *strlcat*( ) function shall return the initial length of the string (if any) pointed to by *dst*, as limited by *dstsize*, plus the length of the string pointed to by *src*; that is, the value that would be returned by *strnlen*(*dst*, *dstsize*) + *strlen*(*src*) before the *strlcat*( ) call.

No return values are reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

The following example detects truncation while combining a path prefix (including trailing <slash>) and a filename to produce a portable pathname:

```
char *prefix, *filenam, pathnam[_POSIX_PATH_MAX];

if (strlcpy(pathnam, prefix, sizeof pathnam) >= sizeof pathnam ||
    strlcat(pathnam, filenam, sizeof pathnam) >= sizeof pathnam)
{
    // truncation occurred
    ...
}
```

This code ensures there is room for the NUL terminator by:

70191 • Calling *strlcpy*( ) with a non-zero *dstsize* argument.

70192 • Only calling *strlcat*( ) if the return value of *strlcpy*( ) indicated that truncation did not occur.

70193 **APPLICATION USAGE**

70194 The return value of the *strlcpy*( ) and *strlcat*( ) functions follows the same convention as
70195 *snprintf*( ); that is, they return the total length of the string they tried to create. If the return value
70196 is greater than or equal to *dstsize*, the output string has been truncated.

70197 **RATIONALE**

70198 None.

70199 **FUTURE DIRECTIONS**

70200 None.

70201 **SEE ALSO**

70202 *fprintf*( ), *strlen*( ), *strncat*( ), *strncpy*( ), *wcslcat*( )

70203 XBD **<string.h>**

70204 **CHANGE HISTORY**

70205 First released in Issue 8.

NAME

strlen, strnlen — get length of fixed size string

SYNOPSIS

```
#include <string.h>

size_t strlen(const char *s);
CX size_t strnlen(const char *s, size_t maxlen);
```

DESCRIPTION

CX For *strlen*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *strlen*( ) function shall compute the number of bytes in the string to which *s* points, not including the terminating NUL character.

CX The *strnlen*( ) function shall compute the smaller of the number of bytes in the array to which *s* points, not including any terminating NUL character, or the value of the *maxlen* argument. The *strnlen*( ) function shall never examine more than *maxlen* bytes of the array pointed to by *s*.

CX The *strlen*( ) and *strnlen*( ) functions shall not change the setting of *errno* on valid input.

RETURN VALUE

The *strlen*( ) function shall return the length of *s*; no return value shall be reserved to indicate an error.

CX The *strnlen*( ) function shall return the number of bytes preceding the first null byte in the array to which *s* points, if *s* contains a null byte within the first *maxlen* bytes; otherwise, it shall return *maxlen*.

ERRORS

No errors are defined.

EXAMPLES

Getting String Lengths

The following example sets the maximum length of *key* and *data* by using *strlen*( ) to get the lengths of those strings.

```
#include <string.h>
...
struct element {
    char *key;
    char *data;
};
...
char *key, *data;
int len;

*keylength = *datalength = 0;
...
if ((len = strlen(key)) > *keylength)
    *keylength = len;
if ((len = strlen(data)) > *datalength)
    *datalength = len;
...
```


APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strlcat( ), *wcslen*( )

XBD **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The RETURN VALUE section is updated to indicate that *strlen*( ) returns the length of *s*, and not *s* itself as was previously stated.

Issue 7

The *strnlen*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0344 [560] is applied.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *strlen*( ) and *strnlen*( ) do not change the setting of *errno* on valid input.

Austin Group Defect 986 is applied, adding *strlcat*( ) to the SEE ALSO section.

70272 **NAME**

70273 strncasecmp, strncasecmp_l — case-insensitive string comparisons

70274 **SYNOPSIS**

70275 ```
#include <strings.h>
```

70276 ```
int strncasecmp(const char *s1, const char *s2, size_t n);
```
70277 ```
int strncasecmp_l(const char *s1, const char *s2,
```
70278 ```
    size_t n, locale_t locale);
```

70279 **DESCRIPTION**

70280 Refer to *strcasecmp*( ).

NAME

strncat — concatenate a string with part of another

SYNOPSIS

```
#include <string.h>

char *strncat(char *restrict s1, const char *restrict s2, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *strncat*( ) function shall append not more than *n* bytes (a NUL character and bytes that follow it are not appended) from the array pointed to by *s2* to the end of the string pointed to by *s1*. The initial byte of *s2* overwrites the NUL character at the end of *s1*. A terminating NUL character is always appended to the result. If copying takes place between objects that overlap, the behavior is undefined.

CX The *strncat*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

The *strncat*( ) function shall return *s1*; no return value shall be reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcat( ), *strlcat*( )

XBD **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The *strncat*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *strncat*( ) does not change the setting of *errno* on valid input.

Austin Group Defect 986 is applied, adding *strlcat*( ) to the SEE ALSO section.

NAME

strncmp — compare part of two strings

SYNOPSIS

```
#include <string.h>

int strncmp(const char *s1, const char *s2, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *strncmp*( ) function shall compare not more than *n* bytes (bytes that follow a NUL character are not compared) from the array pointed to by *s1* to the array pointed to by *s2*.

The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings being compared.

CX The *strncmp*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

Upon successful completion, *strncmp*( ) shall return an integer greater than, equal to, or less than 0, if the possibly null-terminated array pointed to by *s1* is greater than, equal to, or less than the possibly null-terminated array pointed to by *s2* respectively.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcmp( )

XBD **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *strncmp*( ) does not change the setting of *errno* on valid input.

70358 **NAME**

70359 stpncpy, strncpy — copy fixed length string, returning a pointer to the array end

70360 **SYNOPSIS**

70361
```
#include <string.h>
```

70362 CX
```
char *stpncpy(char *restrict s1, const char *restrict s2, size_t n);
```
70363
```
char *strncpy(char *restrict s1, const char *restrict s2, size_t n);
```

70364 **DESCRIPTION**

70365 CX For *strncpy*( ): The functionality described on this reference page is aligned with the ISO C
70366 standard. Any conflict between the requirements described here and the ISO C standard is
70367 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

70368 CX The *stpncpy*( ) and *strncpy*( ) functions shall copy not more than *n* bytes (bytes that follow a NUL
70369 character are not copied) from the array pointed to by *s2* to the array pointed to by *s1*.

70370 If the array pointed to by *s2* is a string that is shorter than *n* bytes, NUL characters shall be
70371 appended to the copy in the array pointed to by *s1*, until *n* bytes in all are written.

70372 If copying takes place between objects that overlap, the behavior is undefined.

70373 CX The *strncpy*( ) and *stpncpy*( ) functions shall not change the setting of *errno* on valid input.

70374 **RETURN VALUE**

70375 CX If a NUL character is written to the destination, the *stpncpy*( ) function shall return the address of
70376 the first such NUL character. Otherwise, it shall return *&s1*[*n*].

70377 The *strncpy*( ) function shall return *s1*.

70378 No return values are reserved to indicate an error.

70379 **ERRORS**

70380 No errors are defined.

70381 **EXAMPLES**

70382 None.

70383 **APPLICATION USAGE**

70384 Applications must provide the space in *s1* for the *n* bytes to be transferred, as well as ensure that
70385 the *s2* and *s1* arrays do not overlap.

70386 Character movement is performed differently in different implementations. Thus, overlapping
70387 moves may yield surprises.

70388 If there is no NUL character byte in the first *n* bytes of the array pointed to by *s2*, the result is not
70389 null-terminated.

70390 **RATIONALE**

70391 None.

70392 **FUTURE DIRECTIONS**

70393 None.

70394 **SEE ALSO**

70395 *strcpy*( ), *strlcat*( ), *wcsncpy*( )

70396 XBD **<string.h>**

70397 **CHANGE HISTORY**

70398 First released in Issue 1. Derived from Issue 1 of the SVID.

70399 **Issue 6**

70400 The *strncpy*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

70401 **Issue 7**

70402 The *stpncpy*( ) function is added from The Open Group Technical Standard, 2006, Extended API
70403 Set Part 1.

70404 **Issue 8**

70405 Austin Group Defect 448 is applied, adding a requirement that *strncpy*( ) and *stpncpy*( ) do not
70406 change the setting of *errno* on valid input.

70407 Austin Group Defect 986 is applied, adding *strlcat*( ) to the SEE ALSO section.

NAME

strndup — duplicate a specific number of bytes from a string

SYNOPSIS

CX
```
#include <string.h>

char *strndup(const char *s, size_t size);
```

DESCRIPTION

Refer to *strdup( )*.

NAME

strnlen — get length of fixed size string

SYNOPSIS

CX
```
#include <string.h>

size_t strnlen(const char *s, size_t maxlen);
```

DESCRIPTION

Refer to *strlen( )*.

NAME

strpbrk — scan a string for a byte

SYNOPSIS

```
#include <string.h>

char *strpbrk(const char *s1, const char *s2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *strpbrk*( ) function shall locate the first occurrence in the string pointed to by *s1* of any byte from the string pointed to by *s2*.

CX The *strpbrk*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

Upon successful completion, *strpbrk*( ) shall return a pointer to the byte or a null pointer if no byte from *s2* occurs in *s1*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strchr( ), *strrchr*( )

XBD **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *strpbrk*( ) does not change the setting of *errno* on valid input.

NAME

strptime — date and time conversion

SYNOPSIS

XSI

```
#include <time.h>

char *strptime(const char *restrict buf, const char *restrict format,
    struct tm *restrict tm);
```

DESCRIPTION

The *strptime*( ) function shall convert the character string pointed to by *buf* to values which are stored in the **tm** structure pointed to by *tm*, using the format specified by *format*.

The application shall ensure that the format is a character string, beginning and ending in its initial shift state, if any. The format is composed of zero or more directives. Each directive is composed of one of the following: one or more white-space bytes; an ordinary character (neither '%' nor a white-space byte); or a conversion specification.

Each conversion specification is introduced by the '%' character after which the following appear in sequence:

- An optional flag, the zero character ('0') or the <plus-sign> character ('+'), which is ignored.
- An optional field width. If a field width is specified, it shall be interpreted as a string of decimal digits that determine the maximum number of bytes converted for the conversion rather than the number of bytes specified below in the description of the conversion specifiers.
- An optional E or O modifier.
- A terminating conversion specifier character that indicates the type of conversion to be applied.

The conversions are determined using the *LC_TIME* category of the current locale. The application shall ensure that there are white-space bytes or other non-alphanumeric bytes between any two conversion specifications unless all of the adjacent conversion specifications convert a known, fixed number of characters. In the following list, the maximum number of characters scanned (excluding the one matching the next directive) is as follows:

- If a maximum field width is specified, then that number
- Otherwise, the pattern "{x}" indicates that the maximum is *x*
- Otherwise, the pattern "[x,y]" indicates that the value shall fall within the range given (both bounds being inclusive), and the maximum number of characters scanned shall be the maximum required to represent any value in the range without leading zeros and without a leading <plus-sign>

The following conversion specifiers are supported.

The results are unspecified if a modifier is specified with a flag or with a minimum field width, or if a field width is specified for any conversion specifier other than C, F, G, Y, or Z.

a The day of the week, using the locale's weekday names; either the abbreviated or full name can be specified. The *tm_wday* member of the **tm** structure pointed to by *tm* shall be set to the corresponding day of the week number (Sunday=0).

70496 `A` Equivalent to `%a`.

70497 `b` The month, using the locale's month names; either the abbreviated or full version of
70498 either the default or the alternative month name can be specified. The *tm_mon* member
70499 of the **tm** structure pointed to by *tm* shall be set to the corresponding month number.

70500 `B` Equivalent to `%b`.

70501 `c` Replaced by the locale's appropriate date and time representation. The members of the
70502 **tm** structure pointed to by *tm* shall be set as specified for the conversions present in the
70503 locale's **d_t_fmt** value.

70504 `C` All but the last two digits of the year {2}; leading zeros shall be permitted but shall not
70505 be required. A leading `'+'` or `'−'` character shall be permitted before any leading zeros
70506 but shall not be required. The *tm_year* member of the **tm** structure pointed to by *tm*
70507 shall be set to the number formed by appending the last two digits of the year to these
70508 digits, minus 1900. If a `y` conversion is also performed, the last two digits of the year
70509 shall be those processed by the `y` conversion; otherwise, they shall be 00.

70510 `d` The day of the month [01,31]; leading zeros shall be permitted but shall not be required.
70511 The *tm_mday* member of the **tm** structure pointed to by *tm* shall be set to this number.

70512 `D` Equivalent to `%m/%d/%y`.

70513 `e` Equivalent to `%d`.

70514 `F` This specifier is similar to `%Y−%m−%d` where the characters up to the first <hyphen-
70515 minus> separator shall be converted as for `%Y` but with unlimited field width, the
70516 characters between the two <hyphen-minus> separators shall be converted as for `%m`,
70517 and the characters after the last <hyphen-minus> separator shall be converted as for
70518 `%d`. If a field width is specified, each of the `%Y`, `%m`, and `%d` conversions shall not
70519 convert any characters past the overall `%F` field width. The members of the **tm** structure
70520 pointed to by *tm* shall be set as specified for the `Y`, `m`, and `d` conversions.

70521 `g` The last 2 digits of the week-based year (see below) as a decimal number (for example,
70522 77). Leading zeros shall be permitted but shall not be required. A leading `'+'` or `'−'`
70523 character shall be permitted before any leading zeros but shall not be required. The
70524 effect of this year, if any, on the **tm** structure pointed to by *tm* is unspecified.

70525 `G` The week-based year (see below) as a decimal number (for example, 1977). Leading
70526 zeros shall be permitted but shall not be required. A leading `'+'` or `'−'` character shall
70527 be permitted before any leading zeros but shall not be required. The effect of this year,
70528 if any, on the **tm** structure pointed to by *tm* is unspecified.

70529 `h` Equivalent to `%b`.

70530 `H` The hour (24-hour clock) [00,23]; leading zeros shall be permitted but shall not be
70531 required. The *tm_hour* member of the **tm** structure pointed to by *tm* shall be set to this
70532 number.

70533 `I` The hour (12-hour clock) [01,12]; leading zeros shall be permitted but shall not be
70534 required. If a `p` conversion is also performed, the *tm_hour* member of the **tm** structure
70535 pointed to by *tm* shall be set to the hour, by the 24-hour clock, corresponding to the
70536 combined results of the `I` and `p` conversions. If a `p` conversion is not also performed,
70537 the behavior is unspecified.

70538 `j` The day number of the year [001,366]; leading zeros shall be permitted but shall not be
70539 required. The *tm_yday* member of the **tm** structure pointed to by *tm* shall be set to this
70540 number minus 1.

`m` The month number [01,12]; leading zeros shall be permitted but shall not be required. The *tm_mon* member of the **tm** structure pointed to by *tm* shall be set to this number minus 1.

`M` The minute [00,59]; leading zeros shall be permitted but shall not be required. The *tm_min* member of the **tm** structure pointed to by *tm* shall be set to this number.

`n` Any white-space bytes.

`p` The locale's equivalent of a.m. or p.m. If an `I` conversion is also performed, the *tm_hour* member of the **tm** structure pointed to by *tm* shall be set as specified for the `I` conversion; otherwise, the behavior is unspecified.

`r` 12-hour clock time, if the 12-hour format is supported in the locale (see XBD Section 7.3.5, on page 152); in the POSIX locale, this shall be equivalent to `%I:%M:%S %p`. The members of the **tm** structure pointed to by *tm* shall be set as specified for the conversions present in the locale's **t_fmt_ampm** value.

`R` Equivalent to `%H:%M`.

`s` The number of seconds since the Epoch as a decimal number (see XBD Section 4.19, on page 107); leading zeros shall be permitted but shall not be required. The effect of this number, if any, on the **tm** structure pointed to by *tm* is unspecified.

`S` The seconds [00,60]; leading zeros shall be permitted but shall not be required. The *tm_sec* member of the **tm** structure pointed to by *tm* shall be set to this number.

`t` Any white-space bytes.

`T` Equivalent to `%H:%M:%S`.

`u` The weekday as a decimal number [1,7], with 1 representing Monday. The *tm_wday* member of the **tm** structure pointed to by *tm* shall be set to this number modulo 7.

`U` The week number of the year (Sunday as the first day of the week) as a decimal number [00,53]; leading zeros shall be permitted but shall not be required. The effect of this week number, if any, on the **tm** structure pointed to by *tm* is unspecified.

`V` The week number of the week-based year (see below) as a decimal number [01,53]. Leading zeros shall be permitted but shall not be required. The effect of this week number, if any, on the **tm** structure pointed to by *tm* is unspecified.

`w` The weekday as a decimal number [0,6], with 0 representing Sunday. The *tm_wday* member of the **tm** structure pointed to by *tm* shall be set to this number.

`W` The week number of the year (Monday as the first day of the week) as a decimal number [00,53]; leading zeros shall be permitted but shall not be required. The effect of this week number, if any, on the **tm** structure pointed to by *tm* is unspecified.

`x` The date, using the locale's date format. The members of the **tm** structure pointed to by *tm* shall be set as specified for the conversions present in the locale's **d_fmt** value.

`X` The time, using the locale's time format. The members of the **tm** structure pointed to by *tm* shall be set as specified for the conversions present in the locale's **t_fmt** value.

`y` The last two digits of the year; leading zeros shall be permitted but shall not be required. A leading `'+'` or `'−'` character shall be permitted before any leading zeros but shall not be required. If a `C` conversion is not also performed, values in the range [69,99] shall refer to years 1969 to 1999 inclusive and values in the range [00,68] shall refer to years 2000 to 2068 inclusive. If a `C` conversion is also performed, the *tm_year*

70584 member of the **tm** structure pointed to by *tm* shall be set as specified for the `C`
70585 conversion; otherwise, the *tm_year* member shall be set to the calculated year minus
70586 1900.

70587 **Note:** It is expected that in a future version of this standard the default century inferred
70588 from a 2-digit year will change. (This would apply to all commands accepting a
70589 2-digit year as input.)

70590 `Y` The full year {4}; leading zeros shall be permitted but shall not be required. A leading
70591 `'+'` or `'−'` character shall be permitted before any leading zeros but shall not be
70592 required. The *tm_year* member of the **tm** structure pointed to by *tm* shall be set to this
70593 number minus 1900.

70594 `z` The offset from UTC in the ISO 8601-1:2019 standard format (`+hhmm` or `−hhmm`). For
70595 example, `"−0430"` means 4 hours 30 minutes behind UTC (west of Greenwich). The
70596 effect of this offset, if any, on the **tm** structure pointed to by *tm* is unspecified.

70597 `Z` The timezone name. If this name matches the name pointed to by *tzname*[1], and the
70598 names pointed to by *tzname*[0] and *tzname*[1] differ, then the *tm_isdst* member of the **tm**
70599 structure pointed to by *tm* shall be set to 1. Otherwise, if this name matches the name
70600 pointed to by *tzname*[0] then the *tm_isdst* member of the **tm** structure pointed to by *tm*
70601 shall be set to 0. The *tm_zone* and *tm_gmtoff* members of the structure may also be set in
70602 an unspecified manner. Members other than *tm_isdst*, *tm_zone*, and *tm_gmtoff* may be
70603 affected if an `s` conversion is also performed but shall otherwise not be affected.

70604 `%` Replaced by `%`.

70605 **Modified Conversion Specifiers**

70606 Some conversion specifiers can be modified by the `E` and `O` modifier characters to indicate that
70607 an alternative format or specification should be used rather than the one normally used by the
70608 unmodified conversion specifier. If the alternative format or specification does not exist in the
70609 current locale, the behavior shall be as if the unmodified conversion specification were used.

70610 `%Ec` The locale's alternative appropriate date and time representation.

70611 `%EC` The name of the base year (period) in the locale's alternative representation.

70612 `%Ex` The locale's alternative date representation.

70613 `%EX` The locale's alternative time representation.

70614 `%Ey` The offset from `%EC` (year only) in the locale's alternative representation.

70615 `%EY` The full alternative year representation.

70616 `%Ob` Equivalent to `%b`.

70617 `%OB` Equivalent to `%b`.

70618 `%Od` The day of the month using the locale's alternative numeric symbols; leading zeros
70619 shall be permitted but shall not be required.

70620 `%Oe` Equivalent to `%Od`.

70621 `%Oh` Equivalent to `%b`.

70622 `%OH` The hour (24-hour clock) using the locale's alternative numeric symbols.

70623 `%OI` The hour (12-hour clock) using the locale's alternative numeric symbols.

70624 %Om The month using the locale's alternative numeric symbols.

70625 %OM The minutes using the locale's alternative numeric symbols.

70626 %OS The seconds using the locale's alternative numeric symbols.

70627 %OU The week number of the year (Sunday as the first day of the week) using the locale's
70628 alternative numeric symbols.

70629 %OV The same as %V but using the locale's alternative numeric symbols.

70630 %Ow The number of the weekday (Sunday=0) using the locale's alternative numeric
70631 symbols.

70632 %OW The week number of the year (Monday as the first day of the week) using the locale's
70633 alternative numeric symbols.

70634 %Oy The year (offset from %C) using the locale's alternative numeric symbols.

70635 %g, %G, and %V convert values according to the ISO 8601-1:2019 standard week-based year. In
70636 this system, weeks begin on a Monday and week 1 of the week-based year is the week that
70637 includes January 4th, which is also the week that includes the first Thursday of the year, and is
70638 also the first week that contains at least four days in the year. If the first Monday of January is
70639 the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding week-based
70640 year (thus, the string "1998 53 6" with format specifier "%G %V %u" represents Saturday 2nd
70641 January 1999). If December 29th, 30th, or 31st is a Monday, it and any following days are part of
70642 week 1 of the following week-based year (thus, the string "1998 01 2" with format specifier
70643 "%G %V %u" represents Tuesday 30th December 1997).

70644 A conversion specification composed of white-space bytes is executed by scanning input up to
70645 the first non-white-space byte (which remains unscanned), or until no more characters can be
70646 scanned.

70647 A conversion specification that is an ordinary character is executed by scanning the next
70648 character from the buffer. If the character scanned from the buffer differs from the one
70649 comprising the directive, the directive fails, and the differing and subsequent characters remain
70650 unscanned.

70651 A series of conversion specifications composed of %n, %t, white-space bytes, or any combination
70652 is executed by scanning up to the first non-white-space byte (which remains unscanned), or until
70653 no more characters can be scanned.

70654 Any other conversion specification is executed by scanning characters until a character matching
70655 the next directive is scanned, or until no more characters can be scanned. These characters,
70656 except the one matching the next directive, are then compared to the locale values associated
70657 with the conversion specifier. If a match is found, values for the affected **tm** structure members
70658 are set as specified in the description of the conversion specification. Case is ignored when
70659 matching items in *buf* such as month or weekday names. If no match is found, *strptime*( ) fails
70660 and no more characters are scanned.

70661 **RETURN VALUE**

70662 Upon successful completion, *strptime*( ) shall return a pointer to the character following the last
70663 character parsed. Otherwise, a null pointer shall be returned.

70664 **ERRORS**

70665 No errors are defined.

EXAMPLES

Convert a Date-Plus-Time String to Broken-Down Time and Then into Seconds

The following example demonstrates the use of *strptime*( ) to convert a string into broken-down time. The broken-down time is then converted into seconds since the Epoch using *mktime*( ).

```
#include <time.h>
...

struct tm tm;
time_t t;

if (strptime("6 Dec 2001 12:33:45", "%d %b %Y %H:%M:%S", &tm) == NULL)
    /* Handle error */;

printf("year: %d; month: %d; day: %d;\n",
        tm.tm_year, tm.tm_mon, tm.tm_mday);
printf("hour: %d; minute: %d; second: %d\n",
        tm.tm_hour, tm.tm_min, tm.tm_sec);
printf("week day: %d; year day: %d\n", tm.tm_wday, tm.tm_yday);

tm.tm_isdst = -1;      /* Not set by strptime(); tells mktime()
                          to determine whether daylight saving time
                          is in effect */
t = mktime(&tm);
if (t == -1)
    /* Handle error */;
printf("seconds since the Epoch: %ld\n", (long) t);"
```

APPLICATION USAGE

Several ``equivalent to'' formats and the special processing of white-space characters are provided in order to ease the use of identical format strings for *strftime*( ) and *strptime*( ).

It should be noted that dates constructed by the *strftime*( ) function with the `%Y` or `%C%y` conversion specifiers may have values larger than 9 999. If the *strptime*( ) function is used to read such values using `%C%y` or `%Y`, the year values will be truncated to four digits. Applications should use `%+`*w*`%y` or `%+`*x*`Y` with *w* and *x* set large enough to contain the full value of any years that will be printed or scanned.

The effect of the `s` conversion is unspecified because existing implementations differ in behavior. Some do a conversion equivalent to *gmtime*( ), ignoring all available timezone information; some do a conversion equivalent to *localtime*( ), using the same timezone it would use and ignoring any timezone information provided by a `z` or `Z` conversion. Although none has been observed, there may be existing (or future) implementations that use timezone information provided by a `z` or `Z` conversion, although using the latter would not be reliable as timezone names are often ambiguous. Applications that need to convert a seconds since the Epoch value to a **tm** structure should call *gmtime*( ) or *localtime*( ) (or their thread-safe equivalents) directly.

The effect of the `z` conversion is unspecified because existing implementations differ in behavior. Some just use it to set the *tm_gmtoff* member of the **tm** structure; some use the value to adjust the other field members to represent UTC, convert the resulting time to a seconds since the Epoch value, and then convert back to a **tm** structure by the equivalent of *localtime*( ). An application that needs either of these behaviors should perform the necessary processing explicitly itself.

Although the `Z` conversion might be expected to set the *tm_zone* member of the **tm** structure, no existing implementation has been found that sets it. Applications that need it set should set it

explicitly after calling *strptime*( ).

See also the APPLICATION USAGE section in *strftime*( ).

It is unspecified whether multiple calls to *strptime*( ) using the same **tm** structure will update the current contents of the structure or overwrite all contents of the structure. Conforming applications should make a single call to *strptime*( ) with a format and all data needed to completely specify the date and time being converted.

RATIONALE

See the RATIONALE section for *strftime*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

fprintf( ), *fscanf*( ), *strftime*( ), *time*( )

XBD **<time.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

Moved from ENHANCED I18N to BASE.

The [ENOSYS] error is removed.

The exact meaning of the `%y` and `%Oy` specifiers is clarified in the DESCRIPTION.

Issue 6

The Open Group Corrigendum U033/5 is applied. The `%r` specifier description is reworded.

The normative text is updated to avoid use of the term ``must'' for application requirements.

The **restrict** keyword is added to the *strptime*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

The Open Group Corrigendum U047/2 is applied.

The DESCRIPTION is updated to use the terms ``conversion specifier'' and ``conversion specification'' for consistency with *strftime*( ).

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/133 is applied, adding the example to the EXAMPLES section.

Issue 7

Austin Group Interpretation 1003.1-2001 #041 is applied, updating the DESCRIPTION and APPLICATION USAGE sections.

Austin Group Interpretation 1003.1-2001 #163 is applied.

SD5-XSH-ERN-67 is applied, correcting the APPLICATION USAGE to remove the impression that `%Y` is 4-digit years.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0345 [920] and XSH/TC2-2008/0346 [919] are applied.

Issue 8

Austin Group Defect 169 is applied, adding the `s` conversion.

Austin Group Defects 258 and 1166 are applied, adding the `Ob`, `OB`, and `Oh` modified conversions.

Austin Group Defect 879 is applied, adding the `F`, `g`, `G`, `u`, `V`, `z`, and `Z` conversions and the `OV` modified conversion, and adding a statement to the week number conversions that their effect on the **tm** structure pointed to by *tm* is unspecified.

Austin Group Defect 1163 is applied, clarifying the handling of white space in the format string.

Austin Group Defect 1307 is applied, changing the `r` conversion in relation to locales that do not support the 12-hour clock format.

Austin Group Defect 1562 is applied, clarifying that it is the application's responsibility to ensure that the format is a character string, beginning and ending in its initial shift state, if any.

Austin Group Defect 1727 is applied, clarifying which members of the **tm** structure are updated by each conversion.

NAME

strrchr — string scanning operation

SYNOPSIS

```
#include <string.h>

char *strrchr(const char *s, int c);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *strrchr*( ) function shall locate the last occurrence of *c* (converted to a **char**) in the string pointed to by *s*. The terminating NUL character is considered to be part of the string.

CX The *strrchr*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

Upon successful completion, *strrchr*( ) shall return a pointer to the byte or a null pointer if *c* does not occur in the string.

ERRORS

No errors are defined.

EXAMPLES

Finding the Base Name of a File

The following example uses *strrchr*( ) to get a pointer to the base name of a file. The *strrchr*( ) function searches backwards through the name of the file to find the last `'/'` character in *name*. This pointer (plus one) will point to the base name of the file.

```
#include <string.h>
...
const char *name;
char *basename;
...
basename = strrchr(name, '/') + 1;
...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strchr( )

XBD **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

70802 **Issue 8**

70803 Austin Group Defect 448 is applied, adding a requirement that *strrchr*( ) does not change the
70804 setting of *errno* on valid input.

NAME

strsignal — get signal message string

SYNOPSIS

CX
```
#include <string.h>

char *strsignal(int signum);
```

DESCRIPTION

The *strsignal*( ) function shall map the signal number in *signum* to an implementation-defined string and shall return a pointer to it. It shall use the same set of messages as the *psignal*( ) function.

The application shall not modify the string returned. The returned pointer might be invalidated or the string content might be overwritten by a subsequent call to *strsignal*( ) or *setlocale*( ). The returned pointer might also be invalidated if the calling thread is terminated.

The contents of the message strings returned by *strsignal*( ) should be determined by the setting of the *LC_MESSAGES* category in the current locale.

The implementation shall behave as if no function defined in this standard calls *strsignal*( ).

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strsignal*( ), then check *errno*.

The *strsignal*( ) function need not be thread-safe.

RETURN VALUE

Upon successful completion, *strsignal*( ) shall return a pointer to a string. Otherwise, if *signum* is not a valid signal number, the return value is unspecified.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If *signum* is not a valid signal number, some implementations return NULL, while for others the *strsignal*( ) function returns a pointer to a string containing an unspecified message denoting an unknown signal. POSIX.1-2024 leaves this return value unspecified.

FUTURE DIRECTIONS

None.

SEE ALSO

psiginfo( ), *setlocale*( ), *sig2str*( )

XBD **<string.h>**

CHANGE HISTORY

First released in Issue 7.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0609 [75] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0347 [656] is applied.

70845 **Issue 8**

70846 Austin Group Defect 1138 is applied, adding *sig2str*( ) to the SEE ALSO section.

70847 Austin Group Defect 1474 is applied, changing the NAME section.

NAME

strspn — get length of a substring

SYNOPSIS

```
#include <string.h>

size_t strspn(const char *s1, const char *s2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *strspn*( ) function shall compute the length (in bytes) of the maximum initial segment of the string pointed to by *s1* which consists entirely of bytes from the string pointed to by *s2*.

CX The *strspn*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

The *strspn*( ) function shall return the computed length; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcspn( )

XBD **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The RETURN VALUE section is updated to indicate that *strspn*( ) returns the length of *s*, and not *s* itself as was previously stated.

Issue 7

SD5-XSH-ERN-182 is applied.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *strspn*( ) does not change the setting of *errno* on valid input.

NAME

strstr — find a substring

SYNOPSIS

```
#include <string.h>

char *strstr(const char *s1, const char *s2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *strstr*( ) function shall locate the first occurrence in the string pointed to by *s1* of the sequence of bytes (excluding the terminating NUL character) in the string pointed to by *s2*.

CX The *strstr*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

Upon successful completion, *strstr*( ) shall return a pointer to the located string or a null pointer if the string is not found.

If *s2* points to a string with zero length, the function shall return *s1*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

memmem( ), *strchr*( )

XBD **<string.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the ANSI C standard.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *strstr*( ) does not change the setting of *errno* on valid input.

Austin Group Defect 1061 is applied, adding *memmem*( ) to the SEE ALSO section.

NAME

strtod, strtof, strtold — convert a string to a double-precision number

SYNOPSIS

```
#include <stdlib.h>

double strtod(const char *restrict nptr, char **restrict endptr);
float strtof(const char *restrict nptr, char **restrict endptr);
long double strtold(const char *restrict nptr, char **restrict endptr);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall convert the initial portion of the string pointed to by *nptr* to **double**, **float**, and **long double** representation, respectively. First, they decompose the input string into three parts:

1. An initial, possibly empty, sequence of white-space bytes
2. A subject sequence interpreted as a floating-point constant or representing infinity or NaN
3. A final string of one or more unrecognized characters, including the terminating NUL character of the input string

Then they shall attempt to convert the subject sequence to a floating-point number, and return the result.

The expected form of the subject sequence is an optional `'+'` or `'−'` sign, then one of the following:

- A non-empty sequence of decimal digits optionally containing a radix character; then an optional exponent part consisting of the character `'e'` or the character `'E'`, optionally followed by a `'+'` or `'−'` character, and then followed by one or more decimal digits
- A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix character; then an optional binary exponent part consisting of the character `'p'` or the character `'P'`, optionally followed by a `'+'` or `'−'` character, and then followed by one or more decimal digits
- One of INF or INFINITY, ignoring case
- One of NAN or NAN($n\text{-}char\text{-}sequence_{opt}$), ignoring case in the NAN part, where:

```
n-char-sequence:
    digit
    nondigit
    n-char-sequence digit
    n-char-sequence nondigit
```

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space byte, that is of the expected form. The subject sequence contains no characters if the input string is not of the expected form.

If the subject sequence has the expected form for a floating-point number, the sequence of characters starting with the first digit or the decimal-point character (whichever occurs first) shall be interpreted as a floating constant of the C language, except that the radix character shall be used in place of a period, and that if neither an exponent part nor a radix character appears in

a decimal floating-point number, or if a binary exponent part does not appear in a hexadecimal floating-point number, an exponent part of the appropriate type with value zero is assumed to follow the last digit in the string. If the subject sequence begins with a <hyphen-minus>, the sequence shall be interpreted as negated. A character sequence INF or INFINITY shall be interpreted as an infinity, if representable in the return type, else as if it were a floating constant that is too large for the range of the return type. A character sequence NAN or NAN(*n-char-sequence*$_{opt}$) shall be interpreted as a quiet NaN, if supported in the return type, else as if it were a subject sequence part that does not have the expected form; the meaning of the *n*-char sequences is implementation-defined. A pointer to the final string is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

If the subject sequence has the hexadecimal form and FLT_RADIX is a power of 2, the value resulting from the conversion is correctly rounded.

CX The radix character is defined in the current locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a <period> (`'.'`).

CX In other than the C or POSIX locale, additional locale-specific subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion shall be performed; the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

These functions shall not change the setting of *errno* if successful.

Since 0 is returned on error and is also a valid return on success, an application wishing to check for error situations should set *errno* to 0, then call *strtod*( ), *strtof*( ), or *strtold*( ), then check *errno*.

RETURN VALUE

Upon successful completion, these functions shall return the converted value. If no conversion could be performed, 0 shall be returned, and *errno* may be set to [EINVAL].

If the correct value would cause an overflow and default rounding is in effect, ±HUGE_VAL, ±HUGE_VALF, or ±HUGE_VALL shall be returned (according to the sign of the value), and *errno* shall be set to [ERANGE].

If the correct value would cause an underflow, a value whose magnitude is no greater than the CX smallest normalized positive number in the return type shall be returned and *errno* set to [ERANGE].

ERRORS

These functions shall fail if:

[ERANGE] The value to be returned would cause overflow and default rounding is in CX effect or the value to be returned would cause underflow.

These functions may fail if:

CX [EINVAL] No conversion could be performed.

EXAMPLES

None.

APPLICATION USAGE

If the subject sequence has the hexadecimal form and FLT_RADIX is not a power of 2, and the result is not exactly representable, the result should be one of the two numbers in the appropriate internal format that are adjacent to the hexadecimal floating source value, with the extra stipulation that the error should have a correct sign for the current rounding direction.

If the subject sequence has the decimal form and at most DECIMAL_DIG (defined in **<float.h>**) significant digits, the result should be correctly rounded. If the subject sequence *D* has the decimal form and more than DECIMAL_DIG significant digits, consider the two bounding, adjacent decimal strings *L* and *U*, both having DECIMAL_DIG significant digits, such that the values of *L*, *D*, and *U* satisfy *L* <= *D* <= *U*. The result should be one of the (equal or adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current rounding direction, with the extra stipulation that the error with respect to *D* should have a correct sign for the current rounding direction.

The changes to *strtod*( ) introduced by the ISO/IEC 9899:1999 standard can alter the behavior of well-formed applications complying with the ISO/IEC 9899:1990 standard and thus earlier versions of this standard. One such example would be:

```
int
what_kind_of_number (char *s)
{
    char *endp;
    double d;
    long l;

    d = strtod(s, &endp);
    if (s != endp && *endp == '\0')
        printf("It's a float with value %g\n", d);
    else
    {
        l = strtol(s, &endp, 0);
        if (s != endp && *endp == '\0')
            printf("It's an integer with value %ld\n", l);
        else
            return 1;
    }
    return 0;
}
```

If the function is called with:

```
what_kind_of_number ("0x10")
```

an ISO/IEC 9899:1990 standard-compliant library will result in the function printing:

```
It's an integer with value 16
```

With the ISO/IEC 9899:1999 standard, the result is:

```
It's a float with value 16
```

The change in behavior is due to the inclusion of floating-point numbers in hexadecimal notation without requiring that either a decimal point or the binary exponent be present.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fscanf( ), *isspace*( ), *localeconv*( ), *setlocale*( ), *strtol*( )

XBD Chapter 7 (on page 127), **<float.h>**, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *strtod*( ) function is updated.
- The *strtof*( ) and *strtold*( ) functions are added.
- The DESCRIPTION is extensively revised.

ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/61 is applied, correcting the second paragraph in the RETURN VALUE section. This change clarifies the sign of the return value.

Issue 7

Austin Group Interpretation 1003.1-2001 #015 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0610 [302], XSH/TC1-2008/0611 [94], and XSH/TC1-2008/0612 [105] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0348 [584] and XSH/TC2-2008/0349 [796] are applied.

Issue 8

Austin Group Defect 1163 is applied, clarifying the handling of white space in the input string.

Austin Group Defect 1213 is applied, correcting some typographic errors in the APPLICATION USAGE section.

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

Austin Group Defect 1686 is applied, adding CX shading to some text in the RETURN VALUE section.

NAME

strtoimax, strtoumax — convert string to integer type

SYNOPSIS

```
#include <inttypes.h>

intmax_t strtoimax(const char *restrict nptr, char **restrict endptr,
    int base);
uintmax_t strtoumax(const char *restrict nptr, char **restrict endptr,
    int base);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall be equivalent to the *strtol*( ), *strtoll*( ), *strtoul*( ), and *strtoull*( ) functions, except that the initial portion of the string shall be converted to **intmax_t** and **uintmax_t** representation, respectively.

RETURN VALUE

These functions shall return the converted value, if any.

CX If no conversion could be performed, zero shall be returned and *errno* may be set to [EINVAL].

CX If the value of *base* is not supported, 0 shall be returned and *errno* shall be set to [EINVAL].

If the correct value is outside the range of representable values, {INTMAX_MAX}, {INTMAX_MIN}, or {UINTMAX_MAX} shall be returned (according to the return type and sign of the value, if any), and *errno* shall be set to [ERANGE].

ERRORS

These functions shall fail if:

CX [EINVAL] The value of *base* is not supported.

[ERANGE] The value to be returned is not representable.

These functions may fail if:

[EINVAL] No conversion could be performed.

EXAMPLES

None.

APPLICATION USAGE

Since the value of **endptr* is unspecified if the value of *base* is not supported, applications should either ensure that *base* has a supported value (0 or between 2 and 36) before the call, or check for an [EINVAL] error before examining **endptr*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strtol( ), *strtoul*( )

XBD **<inttypes.h>**

71127 **CHANGE HISTORY**

71128 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

71129 **Issue 7**

71130 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0613 [453] and XSH/TC1-2008/0614
71131 [453] are applied.

NAME

strtok, strtok_r — split string into tokens

SYNOPSIS

```
#include <string.h>

char *strtok(char *restrict s, const char *restrict sep);
CX char *strtok_r(char *restrict s, const char *restrict sep,
        char **restrict state);
```

DESCRIPTION

CX For *strtok*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

A sequence of calls to *strtok*( ) breaks the string pointed to by *s* into a sequence of tokens, each of which is delimited by a byte from the string pointed to by *sep*. The first call in the sequence has *s* as its first argument, and is followed by calls with a null pointer as their first argument. The separator string pointed to by *sep* may be different from call to call.

The first call in the sequence searches the string pointed to by *s* for the first byte that is *not* contained in the current separator string pointed to by *sep*. If no such byte is found, then there are no tokens in the string pointed to by *s* and *strtok*( ) shall return a null pointer. If such a byte is found, it is the start of the first token.

The *strtok*( ) function then searches from there for a byte that *is* contained in the current separator string. If no such byte is found, the current token extends to the end of the string pointed to by *s*, and subsequent searches for a token shall return a null pointer. If such a byte is found, it is overwritten by a NUL character, which terminates the current token. The *strtok*( ) function saves a pointer to the following byte, from which the next search for a token shall start.

Each subsequent call, with a null pointer as the value of the first argument, starts searching from the saved pointer and behaves as described above.

The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls *strtok*( ).

The *strtok*( ) function need not be thread-safe; however, *strtok*( ) shall avoid data races with all other functions.

CX The *strtok_r*( ) function shall be equivalent to *strtok*( ), except that *strtok_r*( ) shall be thread-safe and the argument *state* points to a user-provided pointer that allows *strtok_r*( ) to maintain state between calls which scan the same string. The application shall ensure that the pointer pointed to by *state* is unique for each string (*s*) being processed concurrently by *strtok_r*( ) calls. The application need not initialize the pointer pointed to by *state* to any particular value. The implementation shall not update the pointer pointed to by *state* to point (directly or indirectly) to resources, other than within the string *s*, that need to be freed or released by the caller.

CX The *strtok*( ) and *strtok_r*( ) functions shall not change the setting of *errno* on valid input.

RETURN VALUE

Upon successful completion, *strtok*( ) shall return a pointer to the first byte of a token. Otherwise, if there is no token, *strtok*( ) shall return a null pointer.

CX The *strtok_r*( ) function shall return a pointer to the token found, or a null pointer when no token is found.

71175 **ERRORS**

71176 No errors are defined.

71177 **EXAMPLES**

71178 **Searching for Word Separators**

71179 The following example searches for tokens separated by <space> characters.

```
71180 #include <string.h>
71181 ...
71182 char *token;
71183 char line[] = "LINE TO BE SEPARATED";
71184 char *search = " ";
71185 /* Token will point to "LINE". */
71186 token = strtok(line, search);
71187 /* Token will point to "TO". */
71188 token = strtok(NULL, search);
```

71189 **Find First two Fields in a Buffer**

71190 The following example uses *strtok*( ) to find two character strings (a key and data associated with
71191 that key) separated by any combination of <space>, <tab>, or <newline> characters at the start
71192 of the array of characters pointed to by *buffer*.

```
71193 #include <string.h>
71194 ...
71195 char    *buffer;
71196 ...
71197 struct element {
71198     char *key;
71199     char *data;
71200 } e;
71201 ...
71202 // Load the buffer...
71203 ...
71204 // Get the key and its data...
71205 e.key = strtok(buffer, " \t\n");
71206 e.data = strtok(NULL, " \t\n");
71207 // Process the rest of the contents of the buffer...
71208 ...
```

71209 **APPLICATION USAGE**

71210 Note that if *sep* is the empty string, *strtok*( ) and *strtok_r*( ) return a pointer to the remainder of the
71211 string being tokenized.

71212 The *strtok_r*( ) function is thread-safe and stores its state in a user-supplied buffer instead of
71213 possibly using a static data area that may be overwritten by an unrelated call from another
71214 thread.

71215 **RATIONALE**

71216 The *strtok*( ) function searches for a separator string within a larger string. It returns a pointer to
71217 the last substring between separator strings. This function uses static storage to keep track of
71218 the current string position between calls. The new function, *strtok_r*( ), takes an additional

argument, *state*, to keep track of the current position in the string.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The *strtok_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *strtok*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

Extensions beyond the ISO C standard are marked.

The *strtok_r*( ) function is marked as part of the Thread-Safe Functions option.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

The **restrict** keyword is added to the *strtok*( ) and *strtok_r*( ) prototypes for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XSH-ERN-235 is applied, correcting an example.

The *strtok_r*( ) function is moved from the Thread-Safe Functions option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0615 [177] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0350 [878] is applied.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *strtok*( ) and *strtok_r*( ) do not change the setting of *errno* on valid input.

Austin Group Defect 1302 is applied, aligning the *strtok*( ) function with the ISO/IEC 9899: 2018 standard.

NAME

strtol, strtoll — convert a string to a long integer

SYNOPSIS

```
#include <stdlib.h>

long strtol(const char *restrict nptr, char **restrict endptr, int base);
long long strtoll(const char *restrict nptr, char **restrict endptr,
    int base)
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall convert the initial portion of the string pointed to by *nptr* to a type **long** and **long long** representation, respectively. First, they decompose the input string into three parts:

1. An initial, possibly empty, sequence of white-space bytes
2. A subject sequence interpreted as an integer represented in some radix determined by the value of *base*
3. A final string of one or more unrecognized characters, including the terminating NUL character of the input string.

Then they shall attempt to convert the subject sequence to an integer, and return the result.

If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant, or hexadecimal constant, any of which may be preceded by a `'+'` or `'−'` sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix `'0'` optionally followed by a sequence of the digits `'0'` to `'7'` only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters `'a'` (or `'A'`) to `'f'` (or `'F'`) with values 10 to 15 respectively.

If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by *base*, optionally preceded by a `'+'` or `'−'` sign. The letters from `'a'` (or `'A'`) to `'z'` (or `'Z'`) inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space byte, that is of the expected form. The subject sequence shall contain no characters if the input string is empty or consists entirely of white-space bytes, or if the first non-white-space byte is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of *base* is 0, the sequence of characters starting with the first digit shall be interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a <hyphen-minus>, the resulting value shall be the negative of the converted value. A pointer to the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

CX In other than the C or POSIX locale, additional locale-specific subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

These functions shall not change the setting of *errno* if successful.

Since 0, {LONG_MIN} or {LLONG_MIN}, and {LONG_MAX} or {LLONG_MAX} are returned on error and are also valid returns on success, an application wishing to check for error situations should set *errno* to 0, then call *strtol*( ) or *strtoll*( ), then check *errno*.

RETURN VALUE

Upon successful completion, these functions shall return the converted value, if any. If no conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

CX If the value of *base* is not supported, 0 shall be returned and *errno* shall be set to [EINVAL].

If the correct value is outside the range of representable values, {LONG_MIN}, {LONG_MAX}, {LLONG_MIN}, or {LLONG_MAX} shall be returned (according to the sign of the value), and *errno* set to [ERANGE].

ERRORS

These functions shall fail if:

CX [EINVAL] The value of *base* is not supported.

[ERANGE] The value to be returned is not representable.

These functions may fail if:

[EINVAL] No conversion could be performed.

EXAMPLES

None.

APPLICATION USAGE

Since the value of **endptr* is unspecified if the value of *base* is not supported, applications should either ensure that *base* has a supported value (0 or between 2 and 36) before the call, or check for an [EINVAL] error before examining **endptr*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fscanf( ), *isalpha*( ), *strtod*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

71334 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
71335 added if no conversion could be performed.

71336 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

71337 • The *strtol*( ) prototype is updated.

71338 • The *strtoll*( ) function is added.

71339 **Issue 7**

71340 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0616 [453], XSH/TC1-2008/0617 [105],
71341 XSH/TC1-2008/0618 [453], XSH/TC1-2008/0619 [453], and XSH/TC1-2008/0620 [453] are
71342 applied.

71343 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0351 [892], XSH/TC2-2008/0352 [584],
71344 XSH/TC2-2008/0353 [796], and XSH/TC2-2008/0354 [892] are applied.

71345 **Issue 8**

71346 Austin Group Defect 700 is applied, clarifying how a subject sequence beginning with <hyphen-
71347 minus> is converted.

71348 Austin Group Defect 1163 is applied, clarifying the handling of white space in the input string.

71349 **NAME**

71350 strtold — convert a string to a double-precision number

71351 **SYNOPSIS**

71352 `#include <stdlib.h>`

71353 `long double strtold(const char *restrict nptr, char **restrict endptr);`

71354 **DESCRIPTION**

71355 Refer to *strtod( )*.

71356 **NAME**

71357 strtoll — convert a string to a long integer

71358 **SYNOPSIS**

71359 ```
#include <stdlib.h>
```

71360 ```
long long strtoll(const char *restrict str, char **restrict endptr,
71361     int base);
```

71362 **DESCRIPTION**

71363 Refer to *strtol( )*.

NAME

strtoul, strtoull — convert a string to an unsigned long

SYNOPSIS

```
#include <stdlib.h>

unsigned long strtoul(const char *restrict str,
    char **restrict endptr, int base);
unsigned long long strtoull(const char *restrict str,
    char **restrict endptr, int base);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall convert the initial portion of the string pointed to by *str* to a type **unsigned long** and **unsigned long long** representation, respectively. First, they decompose the input string into three parts:

1. An initial, possibly empty, sequence of white-space bytes
2. A subject sequence interpreted as an integer represented in some radix determined by the value of *base*
3. A final string of one or more unrecognized characters, including the terminating NUL character of the input string

Then they shall attempt to convert the subject sequence to an unsigned integer, and return the result.

If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant, or hexadecimal constant, any of which may be preceded by a `'+'` or `'−'` sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix `'0'` optionally followed by a sequence of the digits `'0'` to `'7'` only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters `'a'` (or `'A'`) to `'f'` (or `'F'`) with values 10 to 15 respectively.

If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by *base*, optionally preceded by a `'+'` or `'−'` sign. The letters from `'a'` (or `'A'`) to `'z'` (or `'Z'`) inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space byte, that is of the expected form. The subject sequence shall contain no characters if the input string is empty or consists entirely of white-space bytes, or if the first non-white-space byte is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of *base* is 0, the sequence of characters starting with the first digit shall be interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a <hyphen-minus>, the resulting value shall be the negative of the converted value; this action shall be performed in the return type. A pointer to the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

71409 CX In other than the C or POSIX locale, additional locale-specific subject sequence forms may be
71410 accepted.

71411 If the subject sequence is empty or does not have the expected form, no conversion shall be
71412 performed; the value of *str* shall be stored in the object pointed to by *endptr*, provided that *endptr*
71413 is not a null pointer.

71414 These functions shall not change the setting of *errno* if successful.

71415 Since 0, {ULONG_MAX}, and {ULLONG_MAX} are returned on error and are also valid returns
71416 on success, an application wishing to check for error situations should set *errno* to 0, then call
71417 *strtoul*( ) or *strtoull*( ), then check *errno*.

71418 **RETURN VALUE**

71419 Upon successful completion, these functions shall return the converted value, if any. If no
71420 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

71421 CX If the value of *base* is not supported, 0 shall be returned and *errno* shall be set to [EINVAL].

71422 If the correct value is outside the range of representable values, {ULONG_MAX} or
71423 {ULLONG_MAX} shall be returned and *errno* set to [ERANGE].

71424 **ERRORS**

71425 These functions shall fail if:

71426 CX [EINVAL] The value of *base* is not supported.

71427 [ERANGE] The value to be returned is not representable.

71428 These functions may fail if:

71429 CX [EINVAL] No conversion could be performed.

71430 **EXAMPLES**

71431 None.

71432 **APPLICATION USAGE**

71433 Since the value of **endptr* is unspecified if the value of *base* is not supported, applications should
71434 either ensure that *base* has a supported value (0 or between 2 and 36) before the call, or check for
71435 an [EINVAL] error before examining **endptr*.

71436 **RATIONALE**

71437 None.

71438 **FUTURE DIRECTIONS**

71439 None.

71440 **SEE ALSO**

71441 *fscanf*( ), *isalpha*( ), *strtod*( ), *strtol*( )

71442 XBD **<stdlib.h>**

71443 **CHANGE HISTORY**

71444 First released in Issue 4. Derived from the ANSI C standard.

71445 **Issue 5**

71446 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EINVAL] error condition is added for when the value of *base* is not supported.
- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *strtoul*( ) prototype is updated.
- The *strtoull*( ) function is added.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0621 [105], XSH/TC1-2008/0622 [453], and XSH/TC1-2008/0623 [453] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0355 [584] and XSH/TC2-2008/0356 [796] are applied.

Issue 8

Austin Group Defect 700 is applied, clarifying how a subject sequence beginning with <hyphen-minus> is converted.

Austin Group Defect 1163 is applied, clarifying the handling of white space in the input string.

71466 **NAME**

71467 strtoumax — convert a string to an integer type

71468 **SYNOPSIS**

71469 ```
#include <inttypes.h>
```

71470 ```
uintmax_t strtoumax(const char *restrict nptr, char **restrict endptr,
```

71471 ```
    int base);
```

71472 **DESCRIPTION**

71473 Refer to *strtoimax( )*.

  
```

NAME

strxfrm, strxfrm_l — string transformation

SYNOPSIS

```
#include <string.h>

size_t strxfrm(char *restrict s1, const char *restrict s2, size_t n);
CX size_t strxfrm_l(char *restrict s1, const char *restrict s2,
        size_t n, locale_t locale);
```

DESCRIPTION

CX For *strxfrm*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *strxfrm*( ) and *strxfrm_l*( ) functions shall transform the string pointed to by *s2* and place the resulting string into the array pointed to by *s1*. The transformation is such that if *strcmp*( ) is applied to two transformed strings, it shall return a value greater than, equal to, or less than 0, CX corresponding to the result of *strcoll*( ) or *strcoll_l*( ), respectively, applied to the same two CX original strings with the same locale. No more than *n* bytes are placed into the resulting array pointed to by *s1*, including the terminating NUL character. If *n* is 0, *s1* is permitted to be a null pointer. If copying takes place between objects that overlap, the behavior is undefined.

CX The *strxfrm*( ) and *strxfrm_l*( ) functions shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error, an application wishing to check for error CX situations should set *errno* to 0, then call *strxfrm*( ) or *strxfrm_l*( ), then check *errno*.

CX The behavior is undefined if the *locale* argument to *strxfrm_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX Upon successful completion, *strxfrm*( ) and *strxfrm_l*( ) shall return the length of the transformed string (not including the terminating NUL character). If the value returned is *n* or more, the contents of the array pointed to by *s1* are unspecified.

CX On error, *strxfrm*( ) and *strxfrm_l*( ) may set *errno* but no return value is reserved to indicate an error.

ERRORS

These functions may fail if:

CX [EINVAL] The string pointed to by the *s2* argument contains characters outside the domain of the collating sequence.

EXAMPLES

None.

APPLICATION USAGE

The transformation function is such that two transformed strings can be ordered by *strcmp*( ) as appropriate to collating sequence information in the current locale (category *LC_COLLATE*).

The fact that when *n* is 0 *s1* is permitted to be a null pointer is useful to determine the size of the *s1* array prior to making the transformation.

71514 **RATIONALE**

71515 None.

71516 **FUTURE DIRECTIONS**

71517 None.

71518 **SEE ALSO**

71519 *strcmp*( ), *strcoll*( )

71520 XBD **<string.h>**

71521 **CHANGE HISTORY**

71522 First released in Issue 3. Included for alignment with the ISO C standard.

71523 **Issue 5**

71524 The DESCRIPTION is updated to indicate that *errno* does not change if the function is successful.

71525 **Issue 6**

71526 Extensions beyond the ISO C standard are marked.

71527 The following new requirements on POSIX implementations derive from alignment with the
71528 Single UNIX Specification:

71529 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
71530 added if no conversion could be performed.

71531 The *strxfrm*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

71532 **Issue 7**

71533 The *strxfrm_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API
71534 Set Part 4.

71535 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0624 [283], XSH/TC1-2008/0625 [283],
71536 and XSH/TC1-2008/0626 [302] are applied.

71537 **NAME**

71538 swab — swap bytes

71539 **SYNOPSIS**

71540 XSI
```
#include <unistd.h>

void swab(const void *restrict src, void *restrict dest,
    ssize_t nbytes);
```

71543 **DESCRIPTION**

71544 The *swab*( ) function shall copy *nbytes* bytes, which are pointed to by *src*, to the object pointed to
71545 by *dest*, exchanging adjacent bytes. The *nbytes* argument should be even. If *nbytes* is odd, *swab*( )
71546 copies and exchanges *nbytes*–1 bytes and the disposition of the last byte is unspecified. If
71547 copying takes place between objects that overlap, the behavior is undefined. If *nbytes* is
71548 negative, *swab*( ) does nothing.

71549 **RETURN VALUE**

71550 None.

71551 **ERRORS**

71552 No errors are defined.

71553 **EXAMPLES**

71554 None.

71555 **APPLICATION USAGE**

71556 None.

71557 **RATIONALE**

71558 None.

71559 **FUTURE DIRECTIONS**

71560 None.

71561 **SEE ALSO**

71562 XBD **<unistd.h>**

71563 **CHANGE HISTORY**

71564 First released in Issue 1. Derived from Issue 1 of the SVID.

71565 **Issue 6**

71566 The **restrict** keyword is added to the *swab*( ) prototype for alignment with the
71567 ISO/IEC 9899: 1999 standard.

71568 **NAME**

71569 swprintf — print formatted wide-character output

71570 **SYNOPSIS**

71571 ```
#include <stdio.h>
```

71572 ```
#include <wchar.h>
```

71573 ```
int swprintf(wchar_t *restrict ws, size_t n,
```

71574 ```
    const wchar_t *restrict format, ...);
```

71575 **DESCRIPTION**

71576 Refer to *fwprintf( )*.

71577 **NAME**

71578 swscanf — convert formatted wide-character input

71579 **SYNOPSIS**

71580 ```
#include <stdio.h>
```

71581 ```
#include <wchar.h>
```

71582 ```
int swscanf(const wchar_t *restrict ws,
```

71583 ```
    const wchar_t *restrict format, ...);
```

71584 **DESCRIPTION**

71585 Refer to *fwscanf( )*.

71586 **NAME**

71587 symlink, symlinkat — make a symbolic link

71588 **SYNOPSIS**

71589
```
#include <unistd.h>
```

71590
```
int symlink(const char *path1, const char *path2);
```

71591 OH
```
#include <fcntl.h>
```

71592
```
int symlinkat(const char *path1, int fd, const char *path2);
```

71593 **DESCRIPTION**

71594 The *symlink*( ) function shall create a symbolic link called *path2* that contains the string pointed to
71595 by *path1* (*path2* is the name of the symbolic link created, *path1* is the string contained in the
71596 symbolic link).

71597 The string pointed to by *path1* shall be treated only as a string and shall not be validated as a
71598 pathname.

71599 If the *symlink*( ) function fails for any reason other than [EIO], any file named by *path2* shall be
71600 unaffected.

71601 If *path2* names a symbolic link, *symlink*( ) shall fail and set *errno* to [EEXIST].

71602 The symbolic link's user ID shall be set to the process' effective user ID. The symbolic link's
71603 group ID shall be set to the group ID of the parent directory or to the effective group ID of the
71604 process. Implementations shall provide a way to initialize the symbolic link's group ID to the
71605 group ID of the parent directory. Implementations may, but need not, provide an
71606 implementation-defined way to initialize the symbolic link's group ID to the effective group ID
71607 of the calling process.

71608 The values of the file mode bits for the created symbolic link are unspecified. All interfaces
71609 specified by POSIX.1-2024 shall behave as if the contents of symbolic links can always be read,
71610 except that the value of the file mode bits returned in the *st_mode* field of the **stat** structure is
71611 unspecified.

71612 Upon successful completion, *symlink*( ) shall mark for update the last data access, last data
71613 modification, and last file status change timestamps of the symbolic link. Also, the last data
71614 modification and last file status change timestamps of the directory that contains the new entry
71615 shall be marked for update.

71616 The *symlinkat*( ) function shall be equivalent to the *symlink*( ) function except in the case where
71617 *path2* specifies a relative path. In this case the symbolic link is created relative to the directory
71618 associated with the file descriptor *fd* instead of the current working directory. If the access mode
71619 of the open file description associated with the file descriptor is not O_SEARCH, the function
71620 shall check whether directory searches are permitted using the current permissions of the
71621 directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not
71622 perform the check.

71623 If *symlinkat*( ) is passed the special value AT_FDCWD in the *fd* parameter, the current working
71624 directory shall be used and the behavior shall be identical to a call to *symlink*( ).

71625 **RETURN VALUE**

71626 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
71627 return –1 and set *errno* to indicate the error.

71628 **ERRORS**

71629 These functions shall fail if:

71630 71631 71632 [EACCES] Write permission is denied in the directory where the symbolic link is being created, or search permission is denied for a component of the path prefix of *path2*.

71633 [EEXIST] The *path2* argument names an existing file.

71634 71635 [EILSEQ] The last pathname component of *path2* is not a portable filename, and cannot be created in the target directory.

71636 [EIO] An I/O error occurs while reading from or writing to the file system.

71637 71638 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path2* argument.

71639 71640 71641 71642 [ENAMETOOLONG] The length of a component of the pathname specified by the *path2* argument is longer than {NAME_MAX} or the length of the *path1* argument is longer than {SYMLINK_MAX}.

71643 71644 [ENOENT] A component of the path prefix of *path2* does not name an existing file or *path2* is an empty string.

71645 71646 71647 71648 [ENOENT] or [ENOTDIR] The *path2* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters. If *path2* without the trailing <slash> characters would name an existing file, an [ENOENT] error shall not occur.

71649 71650 71651 71652 71653 [ENOSPC] The directory in which the entry for the new symbolic link is being placed cannot be extended because no space is left on the file system containing the directory, or the new symbolic link cannot be created because no space is left on the file system which shall contain the link, or the file system is out of file-allocation resources.

71654 71655 [ENOTDIR] A component of the path prefix of *path2* names an existing file that is neither a directory nor a symbolic link to a directory.

71656 [EROFS] The new symbolic link would reside on a read-only file system.

71657 The *symlinkat*( ) function shall fail if:

71658 71659 71660 [EACCES] The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

71661 71662 [EBADF] The *path2* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

71663 71664 [ENOTDIR] The *path2* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

71665 These functions may fail if:

71666 71667 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path2* argument.

71668 71669 71670 [ENAMETOOLONG] The length of the *path2* argument exceeds {PATH_MAX} or pathname resolution of a symbolic link in the *path2* argument produced an intermediate

result with a length that exceeds {PATH_MAX}.

EXAMPLES

None.

APPLICATION USAGE

Like a hard link, a symbolic link allows a file to have multiple logical names. The presence of a hard link guarantees the existence of a file, even after the original name has been removed. A symbolic link provides no such assurance; in fact, the file named by the *path1* argument need not exist when the link is created. A symbolic link can cross file system boundaries.

Normal permission checks are made on each component of the symbolic link pathname during its resolution.

RATIONALE

The purpose of the *symlinkat*( ) function is to create symbolic links in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *symlink*( ), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *symlinkat*( ) function it can be guaranteed that the created symbolic link is located relative to the desired directory.

Implementations are encouraged to have *symlink*( ) and *symlinkat*( ) report an [EILSEQ] error if the last component of *path2* contains any bytes that have the encoded value of a <newline> character.

FUTURE DIRECTIONS

None.

SEE ALSO

fdopendir( ), *fstatat*( ), *lchown*( ), *link*( ), *open*( ), *readlink*( ), *rename*( ), *unlink*( )

XBD **<fcntl.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The following changes were made to align with the IEEE P1003.1a draft standard:

- The DESCRIPTION text is updated.
- The [ELOOP] optional error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

The *symlinkat*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Additions have been made describing how *symlink*( ) sets the user and group IDs and file mode of the symbolic link, and its effect on timestamps.

Changes are made to allow a directory to be opened for searching.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0627 [146,428], XSH/TC1-2008/0628 [461], XSH/TC1-2008/0629 [146,428], XSH/TC1-2008/0630 [146,428,436], XSH/TC1-2008/0631 [324], XSH/TC1-2008/0632 [278], XSH/TC1-2008/0633 [278], and XSH/TC1-2008/0634 [151] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0357 [873], XSH/TC2-2008/0358 [591], XSH/TC2-2008/0359 [641], XSH/TC2-2008/0360 [817], XSH/TC2-2008/0361 [822], XSH/TC2-2008/0362 [817], and XSH/TC2-2008/0363 [591] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 293 is applied, adding the [EILSEQ] error.

71721 **NAME**

71722 sync — schedule file system updates

71723 **SYNOPSIS**

71724 XSI
```
#include <unistd.h>

void sync(void);
```

71726 **DESCRIPTION**

71727 The *sync*( ) function shall cause all information in memory that updates file systems to be
71728 scheduled for writing out to all file systems.

71729 The writing, although scheduled, is not necessarily complete upon return from *sync*( ).

71730 **RETURN VALUE**

71731 The *sync*( ) function shall not return a value.

71732 **ERRORS**

71733 No errors are defined.

71734 **EXAMPLES**

71735 None.

71736 **APPLICATION USAGE**

71737 None.

71738 **RATIONALE**

71739 None.

71740 **FUTURE DIRECTIONS**

71741 None.

71742 **SEE ALSO**

71743 *fsync*( )

71744 XBD **<unistd.h>**

71745 **CHANGE HISTORY**

71746 First released in Issue 4, Version 2.

71747 **Issue 5**

71748 Moved from X/OPEN UNIX extension to BASE.

71749 **NAME**

71750 sysconf — get configurable system variables

71751 **SYNOPSIS**

71752 `#include <unistd.h>`

71753 `long sysconf(int name);`

71754 **DESCRIPTION**

71755 The *sysconf*( ) function provides a method for the application to determine the current value of a
71756 configurable system limit or option (*variable*). The implementation shall support all of the
71757 variables listed in the following table and may support others.

71758 The *name* argument represents the system variable to be queried. The following table lists the
71759 minimal set of system variables from **<limits.h>** or **<unistd.h>** that can be returned by *sysconf*( ),
71760 and the symbolic constants defined in **<unistd.h>** that are the corresponding values used for
71761 *name*.

71762

Variable	Value of Name
{AIO_LISTIO_MAX}	_SC_AIO_LISTIO_MAX
{AIO_MAX}	_SC_AIO_MAX
{AIO_PRIO_DELTA_MAX}	_SC_AIO_PRIO_DELTA_MAX
{ARG_MAX}	_SC_ARG_MAX
{ATEXIT_MAX}	_SC_ATEXIT_MAX
{BC_BASE_MAX}	_SC_BC_BASE_MAX
{BC_DIM_MAX}	_SC_BC_DIM_MAX
{BC_SCALE_MAX}	_SC_BC_SCALE_MAX
{BC_STRING_MAX}	_SC_BC_STRING_MAX
{CHILD_MAX}	_SC_CHILD_MAX
Clock ticks/second	_SC_CLK_TCK
{COLL_WEIGHTS_MAX}	_SC_COLL_WEIGHTS_MAX
{DELAYTIMER_MAX}	_SC_DELAYTIMER_MAX
{EXPR_NEST_MAX}	_SC_EXPR_NEST_MAX
{HOST_NAME_MAX}	_SC_HOST_NAME_MAX
{IOV_MAX}	_SC_IOV_MAX
{LINE_MAX}	_SC_LINE_MAX
{LOGIN_NAME_MAX}	_SC_LOGIN_NAME_MAX
{NGROUPS_MAX}	_SC_NGROUPS_MAX
Initial size of *getgrgid_r*( ) and *getgrnam_r*( ) data buffers	_SC_GETGR_R_SIZE_MAX
Initial size of *getpwuid_r*( ) and *getpwnam_r*( ) data buffers	_SC_GETPW_R_SIZE_MAX
{MQ_OPEN_MAX}	_SC_MQ_OPEN_MAX
{MQ_PRIO_MAX}	_SC_MQ_PRIO_MAX
Maximum number of execution units that can be made available to run threads†	_SC_NPROCESSORS_CONF
Maximum number of execution units currently available to run threads†	_SC_NPROCESSORS_ONLN
Highest supported signal number +1	_SC_NSIG
{OPEN_MAX}	_SC_OPEN_MAX
{PAGE_SIZE}	_SC_PAGE_SIZE
{PAGESIZE}	_SC_PAGESIZE
{PTHREAD_DESTRUCTOR_ITERATIONS}	_SC_THREAD_DESTRUCTOR_ITERATIONS

Variable	Value of Name
{PTHREAD_KEYS_MAX}	_SC_THREAD_KEYS_MAX
{PTHREAD_STACK_MIN}	_SC_THREAD_STACK_MIN
{PTHREAD_THREADS_MAX}	_SC_THREAD_THREADS_MAX
{RE_DUP_MAX}	_SC_RE_DUP_MAX
{RTSIG_MAX}	_SC_RTSIG_MAX
{SEM_NSEMS_MAX}	_SC_SEM_NSEMS_MAX
{SEM_VALUE_MAX}	_SC_SEM_VALUE_MAX
{SIGQUEUE_MAX}	_SC_SIGQUEUE_MAX
{STREAM_MAX}	_SC_STREAM_MAX
{SYMLOOP_MAX}	_SC_SYMLOOP_MAX
{TIMER_MAX}	_SC_TIMER_MAX
{TTY_NAME_MAX}	_SC_TTY_NAME_MAX
{TZNAME_MAX}	_SC_TZNAME_MAX
_POSIX_ADVISORY_INFO	_SC_ADVISORY_INFO
_POSIX_BARRIERS	_SC_BARRIERS
_POSIX_ASYNCHRONOUS_IO	_SC_ASYNCHRONOUS_IO
_POSIX_CLOCK_SELECTION	_SC_CLOCK_SELECTION
_POSIX_CPUTIME	_SC_CPUTIME
_POSIX_DEVICE_CONTROL	_SC_DEVICE_CONTROL
_POSIX_FSYNC	_SC_FSYNC
_POSIX_IPV6	_SC_IPV6
_POSIX_JOB_CONTROL	_SC_JOB_CONTROL
_POSIX_MAPPED_FILES	_SC_MAPPED_FILES
_POSIX_MEMLOCK	_SC_MEMLOCK
_POSIX_MEMLOCK_RANGE	_SC_MEMLOCK_RANGE
_POSIX_MEMORY_PROTECTION	_SC_MEMORY_PROTECTION
_POSIX_MESSAGE_PASSING	_SC_MESSAGE_PASSING
_POSIX_MONOTONIC_CLOCK	_SC_MONOTONIC_CLOCK
_POSIX_PRIORITIZED_IO	_SC_PRIORITIZED_IO
_POSIX_PRIORITY_SCHEDULING	_SC_PRIORITY_SCHEDULING
_POSIX_RAW_SOCKETS	_SC_RAW_SOCKETS
_POSIX_READER_WRITER_LOCKS	_SC_READER_WRITER_LOCKS
_POSIX_REALTIME_SIGNALS	_SC_REALTIME_SIGNALS
_POSIX_REGEXP	_SC_REGEXP
_POSIX_SAVED_IDS	_SC_SAVED_IDS
_POSIX_SEMAPHORES	_SC_SEMAPHORES
_POSIX_SHARED_MEMORY_OBJECTS	_SC_SHARED_MEMORY_OBJECTS
_POSIX_SHELL	_SC_SHELL
_POSIX_SPAWN	_SC_SPAWN
_POSIX_SPIN_LOCKS	_SC_SPIN_LOCKS
_POSIX_SPORADIC_SERVER	_SC_SPORADIC_SERVER
_POSIX_SS_REPL_MAX	_SC_SS_REPL_MAX
_POSIX_SYNCHRONIZED_IO	_SC_SYNCHRONIZED_IO
_POSIX_THREAD_ATTR_STACKADDR	_SC_THREAD_ATTR_STACKADDR
_POSIX_THREAD_ATTR_STACKSIZE	_SC_THREAD_ATTR_STACKSIZE
_POSIX_THREAD_CPUTIME	_SC_THREAD_CPUTIME
_POSIX_THREAD_PRIO_INHERIT	_SC_THREAD_PRIO_INHERIT
_POSIX_THREAD_PRIO_PROTECT	_SC_THREAD_PRIO_PROTECT
_POSIX_THREAD_PRIORITY_SCHEDULING	_SC_THREAD_PRIORITY_SCHEDULING
_POSIX_THREAD_PROCESS_SHARED	_SC_THREAD_PROCESS_SHARED

Variable	Value of Name
_POSIX_THREAD_ROBUST_PRIO_INHERIT	_SC_THREAD_ROBUST_PRIO_INHERIT
_POSIX_THREAD_ROBUST_PRIO_PROTECT	_SC_THREAD_ROBUST_PRIO_PROTECT
_POSIX_THREAD_SAFE_FUNCTIONS	_SC_THREAD_SAFE_FUNCTIONS
_POSIX_THREAD_SPORADIC_SERVER	_SC_THREAD_SPORADIC_SERVER
_POSIX_THREADS	_SC_THREADS
_POSIX_TIMEOUTS	_SC_TIMEOUTS
_POSIX_TIMERS	_SC_TIMERS
_POSIX_TYPED_MEMORY_OBJECTS	_SC_TYPED_MEMORY_OBJECTS
_POSIX_VERSION	_SC_VERSION
_POSIX_V8_ILP32_OFF32	_SC_V8_ILP32_OFF32
_POSIX_V8_ILP32_OFFBIG	_SC_V8_ILP32_OFFBIG
_POSIX_V8_LP64_OFF64	_SC_V8_LP64_OFF64
_POSIX_V8_LPBIG_OFFBIG	_SC_V8_LPBIG_OFFBIG
OB _POSIX_V7_ILP32_OFF32	_SC_V7_ILP32_OFF32
_POSIX_V7_ILP32_OFFBIG	_SC_V7_ILP32_OFFBIG
_POSIX_V7_LP64_OFF64	_SC_V7_LP64_OFF64
_POSIX_V7_LPBIG_OFFBIG	_SC_V7_LPBIG_OFFBIG
_POSIX2_C_BIND	_SC_2_C_BIND
_POSIX2_C_DEV	_SC_2_C_DEV
_POSIX2_CHAR_TERM	_SC_2_CHAR_TERM
_POSIX2_FORT_RUN	_SC_2_FORT_RUN
_POSIX2_LOCALEDEF	_SC_2_LOCALEDEF
_POSIX2_SW_DEV	_SC_2_SW_DEV
_POSIX2_UPE	_SC_2_UPE
_POSIX2_VERSION	_SC_2_VERSION
_XOPEN_CRYPT	_SC_XOPEN_CRYPT
_XOPEN_ENH_I18N	_SC_XOPEN_ENH_I18N
_XOPEN_REALTIME	_SC_XOPEN_REALTIME
_XOPEN_REALTIME_THREADS	_SC_XOPEN_REALTIME_THREADS
_XOPEN_SHM	_SC_XOPEN_SHM
_XOPEN_UNIX	_SC_XOPEN_UNIX
_XOPEN_UUCP	_SC_XOPEN_UUCP
_XOPEN_VERSION	_SC_XOPEN_VERSION

† The nature of an execution unit and the precise conditions under which an execution unit is considered to be available, or can be made available, or how many threads it can execute in parallel, are implementation-defined.

RETURN VALUE

If *name* is an invalid value, *sysconf*( ) shall return –1 and set *errno* to indicate the error. If the variable corresponding to *name* is described in **<limits.h>** as a maximum or minimum value and the variable has no limit, *sysconf*( ) shall return –1 without changing the value of *errno*. Note that indefinite limits do not imply infinite limits; see **<limits.h>**.

Otherwise, *sysconf*( ) shall return the current variable value on the system. The value returned shall not be more restrictive than the corresponding value described to the application when it was compiled with the implementation's **<limits.h>** or **<unistd.h>**. The value returned for *name* arguments other than _SC_NPROCESSORS_ONLN shall not change during the lifetime of the calling process, except that *sysconf*(_SC_OPEN_MAX) may return different values before and after a call to *setrlimit*( ) which changes the RLIMIT_NOFILE soft limit.

If the variable corresponding to *name* is dependent on an unsupported option, the results are unspecified.

ERRORS

The *sysconf*( ) function shall fail if:

[EINVAL] The value of the *name* argument is invalid.

EXAMPLES

None.

APPLICATION USAGE

As –1 is a permissible return value in a successful situation, an application wishing to check for error situations should set *errno* to 0, then call *sysconf*( ), and, if it returns –1, check to see if *errno* is non-zero.

Application developers should check whether an option, such as _POSIX_SPORADIC_SERVER, is supported prior to obtaining and using values for related variables, such as _POSIX_SS_REPL_MAX.

Although the queries _SC_NPROCESSORS_CONF and _SC_NPROCESSORS_ONLN provide a way for a class of ``heavy-load'' application to estimate the optimal number of threads that can be created to maximize throughput, real-world environments have complications that affect the actual efficiency that can be achieved. For example:

- There may be more than one ``heavy-load'' application running on the system.
- The system may be on battery power, and applications should co-ordinate with the system to ensure that a long-running task can pause, resume, and successfully complete even in the event of a power outage.

In case a portable ``heavy-load'' application wants to avoid the use of extensions, its developers may wish to create threads based on the logical partition of the long-running task, or utilize heuristics such as the ratio between CPU time and real time.

RATIONALE

This functionality was added in response to requirements of application developers and of system vendors who deal with many international system configurations. It is closely related to *pathconf*( ) and *fpathconf*( ).

Although a conforming application can run on all systems by never demanding more resources than the minimum values published in this volume of POSIX.1-2024, it is useful for that application to be able to use the actual value for the quantity of a resource available on any given system. To do this, the application makes use of the value of a symbolic constant in **<limits.h>** or **<unistd.h>**.

However, once compiled, the application must still be able to cope if the amount of resource available is increased. To that end, an application may need a means of determining the quantity of a resource, or the presence of an option, at execution time.

Two examples are offered:

1. Applications may wish to act differently on systems with or without job control. Applications vendors who wish to distribute only a single binary package to all instances of a computer architecture would be forced to assume job control is never available if it were to rely solely on the **<unistd.h>** value published in this volume of POSIX.1-2024.
2. International applications vendors occasionally require knowledge of the number of clock ticks per second. Without these facilities, they would be required to either distribute their applications partially in source form or to have 50 Hz and 60 Hz versions for the various countries in which they operate.

It is the knowledge that many applications are actually distributed widely in executable form that leads to this facility. If limited to the most restrictive values in the headers, such applications would have to be prepared to accept the most limited environments offered by the smallest microcomputers. Although this is entirely portable, there was a consensus that they should be able to take advantage of the facilities offered by large systems, without the restrictions associated with source and object distributions.

During the discussions of this feature, it was pointed out that it is almost always possible for an application to discern what a value might be at runtime by suitably testing the various functions themselves. And, in any event, it could always be written to adequately deal with error returns from the various functions. In the end, it was felt that this imposed an unreasonable level of complication and sophistication on the application developer.

This runtime facility is not meant to provide ever-changing values that applications have to check multiple times. The values are seen as changing no more frequently than once per system initialization, such as by a system administrator or operator with an automatic configuration program. This volume of POSIX.1-2024 specifies that they shall not change within the lifetime of the process.

Some values apply to the system overall and others vary at the file system or directory level. The latter are described in *fpathconf( )*.

Note that all values returned must be expressible as integers. String values were considered, but the additional flexibility of this approach was rejected due to its added complexity of implementation and use.

Some values, such as {PATH_MAX}, are sometimes so large that they must not be used to, say, allocate arrays. The *sysconf*( ) function returns a negative value to show that this symbolic constant is not even defined in this case.

Similar to *pathconf*( ), this permits the implementation not to have a limit. When one resource is infinite, returning an error indicating that some other resource limit has been reached is conforming behavior.

FUTURE DIRECTIONS

None.

SEE ALSO

confstr( ), *fpathconf( )*

XBD **<limits.h>**, **<unistd.h>**

XCU *getconf*

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

The _XBS_ variables and name values are added to the table of system variables in the DESCRIPTION. These are all marked EX.

Issue 6

The symbol CLK_TCK is obsolescent and removed. It is replaced with the phrase ``clock ticks per second''.

The symbol {PASS_MAX} is removed.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Table entries are added for the following variables: _SC_REGEXP, _SC_SHELL, _SC_REGEX_VERSION, _SC_SYMLOOP_MAX.

The following *sysconf*( ) variables and their associated names are added for alignment with IEEE Std 1003.1d-1999:

_POSIX_ADVISORY_INFO
_POSIX_CPUTIME
_POSIX_SPAWN
_POSIX_SPORADIC_SERVER
_POSIX_THREAD_CPUTIME
_POSIX_THREAD_SPORADIC_SERVER
_POSIX_TIMEOUTS

The following changes are made to the DESCRIPTION for alignment with IEEE Std 1003.1j-2000:

- A statement expressing the dependency of support for some system variables on implementation options is added.
- The following system variables are added:

  _POSIX_BARRIERS
  _POSIX_CLOCK_SELECTION
  _POSIX_MONOTONIC_CLOCK
  _POSIX_READER_WRITER_LOCKS
  _POSIX_SPIN_LOCKS
  _POSIX_TYPED_MEMORY_OBJECTS

The following system variables are added for alignment with IEEE Std 1003.2d-1994:

_POSIX2_PBS
_POSIX2_PBS_ACCOUNTING
_POSIX2_PBS_LOCATE
_POSIX2_PBS_MESSAGE
_POSIX2_PBS_TRACK

The following *sysconf*( ) variables and their associated names are added for alignment with IEEE Std 1003.1q-2000:

_POSIX_TRACE
_POSIX_TRACE_EVENT_FILTER
_POSIX_TRACE_INHERIT
_POSIX_TRACE_LOG

The macros associated with the *c89* programming models are marked LEGACY, and new equivalent macros associated with *c99* are introduced.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/62 is applied, updating the DESCRIPTION to denote that the _PC* and _SC* symbols are now required to be supported. A corresponding change has been made in the Base Definitions volume of POSIX.1-2024. The deletion in the second paragraph removes some duplicated text. Additional symbols that were erroneously omitted from this reference page have been added.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/63 is applied, making it clear in the RETURN VALUE section that the value returned for *sysconf*(_SC_OPEN_MAX) may change if a

call to *setrlimit*( ) adjusts the RLIMIT_NOFILE soft limit.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/134 is applied, updating the DESCRIPTION to remove an erroneous entry for _POSIX_SYMLOOP_MAX. This corrects an error in IEEE Std 1003.1-2001/Cor 1-2002.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/135 is applied, removing _POSIX_FILE_LOCKING, _POSIX_MULTI_PROCESS, _POSIX2_C_VERSION, and _XOPEN_XCU_VERSION (and their associated _SC_* variables) from the DESCRIPTION and APPLICATION USAGE sections.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/136 is applied, adding the following constants (and their associated _SC_* variables) to the DESCRIPTION:

_POSIX_SS_REPL_MAX
_POSIX_TRACE_EVENT_NAME_MAX
_POSIX_TRACE_NAME_MAX
_POSIX_TRACE_SYS_MAX
_POSIX_TRACE_USER_EVENT_MAX

The RETURN VALUE and APPLICATION USAGE sections are updated to note that if variables are dependent on unsupported options, the results are unspecified.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/137 is applied, removing _REGEX_VERSION and _SC_REGEX_VERSION.

Issue 7

Austin Group Interpretation 1003.1-2001 #160 is applied.

SD5-XSH-ERN-166 is applied, changing ``Maximum size'' to ``Initial size'' for the ``Maximum size of ...'' entries in the table in the DESCRIPTION.

The variables for the supported programming environments are updated to be V7 and the LEGACY variables are removed.

The following constants are added:

_POSIX_THREAD_ROBUST_PRIO_INHERIT
_POSIX_THREAD_ROBUST_PRIO_PROTECT

The _XOPEN_UUCP variable and its associated _SC_XOPEN_UUCP value is added to the table of system variables.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0364 [752] is applied.

Issue 8

Austin Group Defect 51 is applied, moving the *getrlimit*( ) and *setrlimit*( ) functions, excluding the RLIMIT_CPU and RLIMIT_FSIZE limits, from the XSI option to the Base.

Austin Group Defects 339 and 1608 are applied, adding _SC_NPROCESSORS_CONF and _SC_NPROCESSORS_ONLN.

Austin Group Defect 729 is applied, adding _SC_DEVICE_CONTROL.

Austin Group Defect 741 is applied, adding _SC_NSIG.

Austin Group Defect 1330 is applied, removing obsolescent interfaces and changing ``_V7_'' to ``_V8_'' and ``_V6_'' to ``_V7_''.

72069 **NAME**

72070 syslog — log a message

72071 **SYNOPSIS**

72072 XSI
```
#include <syslog.h>

void syslog(int priority, const char *message, ... /* argument */);
```
72073

72074 **DESCRIPTION**

72075 Refer to *closelog( )*.

72076 **NAME**

72077 system — issue a command

72078 **SYNOPSIS**

72079 `#include <stdlib.h>`

72080 `int system(const char *`*`command`*`);`

72081 **DESCRIPTION**

72082 CX The functionality described on this reference page is aligned with the ISO C standard. Any
72083 conflict between the requirements described here and the ISO C standard is unintentional. This
72084 volume of POSIX.1-2024 defers to the ISO C standard.

72085 If *command* is a null pointer, the *system*( ) function shall determine whether the host environment
72086 has a command processor. If *command* is not a null pointer, the *system*( ) function shall pass the
72087 string pointed to by *command* to that command processor to be executed in an implementation-
72088 defined manner; this might then cause the program calling *system*( ) to behave in a non-
72089 conforming manner or to terminate.

72090 CX The *system*( ) function shall behave as if a child process were created using *fork*( ), and the child
72091 process invoked the *sh* utility using *execl*( ) as follows:

72092 `execl(<`*`shell path`*`>, "sh", "-c", "--",` *`command`*`, (char *)0);`

72093 where **<***shell path***>** is an unspecified pathname for the *sh* utility. It is implementation-defined
72094 whether the handlers registered with *pthread_atfork*( ) are called as part of the creation of the
72095 child process.

72096 The *system*( ) function shall ignore the SIGINT and SIGQUIT signals, and shall block the
72097 SIGCHLD signal, while waiting for the command to terminate. If this might cause the
72098 application to miss a signal that would have killed it, then the application should examine the
72099 return value from *system*( ) and take whatever action is appropriate to the application if the
72100 command terminated due to receipt of a signal.

72101 The *system*( ) function shall not affect the termination status of any child of the calling processes
72102 other than the process or processes it itself creates.

72103 The *system*( ) function shall not return until the child process has terminated.

72104 If concurrent calls to *system*( ) are made from multiple threads, it is unspecified whether:

72105 • each call saves and restores the dispositions of the SIGINT and SIGQUIT signals
72106 independently, or

72107 • in a set of concurrent calls the dispositions in effect after the last call returns are those that
72108 were in effect on entry to the first call.

72109 If a thread is cancelled while it is in a call to *system*( ), it is unspecified whether the child process
72110 is terminated and waited for, or is left running.

72111 **RETURN VALUE**

72112 If *command* is a null pointer, *system*( ) shall return non-zero to indicate that a command processor
72113 CX is available, or zero if none is available. The *system*( ) function shall always return non-zero
72114 when *command* is NULL.

72115 CX If *command* is not a null pointer, *system*( ) shall return the termination status of the command
72116 language interpreter in the format specified by *waitpid*( ). The termination status shall be as
72117 defined for the *sh* utility; otherwise, the termination status is unspecified. If some error prevents
72118 the command language interpreter from executing after the child process is created, the return
72119 value from *system*( ) shall be as if the command language interpreter had terminated using

exit(127) or *_exit*(127). If a child process cannot be created, or if the termination status for the command language interpreter cannot be obtained, *system*( ) shall return –1 and set *errno* to indicate the error.

ERRORS

CX The *system*( ) function may set *errno* values as described by *fork*( ).

In addition, *system*( ) may fail if:

CX [ECHILD] The status of the child process created by *system*( ) is no longer available.

EXAMPLES

None.

APPLICATION USAGE

If the return value of *system*( ) is not –1, its value can be decoded through the use of the macros described in **<sys/wait.h>**. For convenience, these macros are also provided in **<stdlib.h>**.

Note that, while *system*( ) must ignore SIGINT and SIGQUIT and block SIGCHLD while waiting for the child to terminate, the handling of signals in the executed command is as specified by *fork*( ) and *exec*. For example, if SIGINT is being caught or is set to SIG_DFL when *system*( ) is called, then the child is started with SIGINT handling set to SIG_DFL.

Ignoring SIGINT and SIGQUIT in the parent process prevents coordination problems (two processes reading from the same terminal, for example) when the executed command ignores or catches one of the signals. It is also usually the correct action when the user has given a command to the application to be executed synchronously (as in the `'!'` command in many interactive applications). In either case, the signal should be delivered only to the child process, not to the application itself. There is one situation where ignoring the signals might have less than the desired effect. This is when the application uses *system*( ) to perform some task invisible to the user. If the user typed the interrupt character (`"^C"`, for example) while *system*( ) is being used in this way, one would expect the application to be killed, but only the executed command is killed. Applications that use *system*( ) in this way should carefully check the return status from *system*( ) to see if the executed command was successful, and should take appropriate action when the command fails.

Blocking SIGCHLD while waiting for the child to terminate prevents the application from catching the signal and obtaining status from *system*( )'s child process before *system*( ) can get the status itself.

The context in which the utility is ultimately executed may differ from that in which *system*( ) was called. For example, file descriptors that have the FD_CLOEXEC or FD_CLOFORK flag set are closed, and the process ID and parent process ID are different. Also, if the executed utility changes its environment variables or its current working directory, that change is not reflected in the caller's context.

There is no defined way for an application to find the specific path for the shell. However, *confstr*( ) can provide a value for *PATH* that is guaranteed to find the *sh* utility.

Although *system*( ) is required to be thread-safe, it is recommended that concurrent calls from multiple threads are avoided, since *system*( ) is not required to coordinate the saving and restoring of the dispositions of the SIGINT and SIGQUIT signals across a set of overlapping calls, and therefore the signals might end up being set to ignored after the last call returns. Applications should also avoid cancelling a thread while it is in a call to *system*( ) as the child process may be left running in that event. In addition, if another thread alters the disposition of the SIGCHLD signal, a call to *signal*( ) may produce unexpected results.

RATIONALE

The *system*( ) function should not be used by programs that have set user (or group) ID privileges. The *fork*( ) and *exec* family of functions (except *execlp*( ) and *execvp*( )), should be used instead. This prevents any unforeseen manipulation of the environment of the user that could cause execution of commands not anticipated by the calling program.

There are three levels of specification for the *system*( ) function. The ISO C standard gives the most basic. It requires that the function exists, and defines a way for an application to query whether a command language interpreter exists. It says nothing about the command language or the environment in which the command is interpreted.

POSIX.1-2024 places additional restrictions on *system*( ). It requires that if there is a command language interpreter, the environment must be as specified by *fork*( ) and *exec*. This ensures, for example, that close-on-*exec* works, that process-owned file locks are not inherited, and that the process ID is different. It also specifies the return value from *system*( ) when the command line can be run, thus giving the application some information about the command's completion status.

Finally, POSIX.1-2024 requires the command to be interpreted as in the shell command language defined in the Shell and Utilities volume of POSIX.1-2024.

Note that, *system*(NULL) is required to return non-zero, indicating that there is a command language interpreter. At first glance, this would seem to conflict with the ISO C standard which allows *system*(NULL) to return zero. There is no conflict, however. A system must have a command language interpreter, and is non-conforming if none is present. It is therefore permissible for the *system*( ) function on such a system to implement the behavior specified by the ISO C standard as long as it is understood that the implementation does not conform to POSIX.1-2024 if *system*(NULL) returns zero.

It was explicitly decided that when *command* is NULL, *system*( ) should not be required to check to make sure that the command language interpreter actually exists with the correct mode, that there are enough processes to execute it, and so on. The call *system*(NULL) could, theoretically, check for such problems as too many existing child processes, and return zero. However, it would be inappropriate to return zero due to such a (presumably) transient condition. If some condition exists that is not under the control of this application and that would cause any *system*( ) call to fail, that system has been rendered non-conforming.

Early drafts required, or allowed, *system*( ) to return with *errno* set to [EINTR] if it was interrupted with a signal. This error return was removed, and a requirement that *system*( ) not return until the child has terminated was added. This means that if a *waitpid*( ) call in *system*( ) exits with *errno* set to [EINTR], *system*( ) must reissue the *waitpid*( ). This change was made for two reasons:

1. There is no way for an application to clean up if *system*( ) returns [EINTR], short of calling *wait*( ), and that could have the undesirable effect of returning the status of children other than the one started by *system*( ).
2. While it might require a change in some historical implementations, those implementations already have to be changed because they use *wait*( ) instead of *waitpid*( ).

Note that if the application is catching SIGCHLD signals, it will receive such a signal before a successful *system*( ) call returns.

To conform to POSIX.1-2024, *system*( ) must use *waitpid*( ), or some similar function, instead of *wait*( ).

The following code sample illustrates how *system*( ) might be implemented on an

implementation conforming to POSIX.1-2024.

```
int system(const char *cmd)
{
    int stat;
    pid_t pid;
    struct sigaction sa, savintr, savequit;
    sigset_t saveblock;
    if (cmd == NULL)
        return(1);
    sa.sa_handler = SIG_IGN;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sigemptyset(&savintr.sa_mask);
    sigemptyset(&savequit.sa_mask);
    sigaction(SIGINT, &sa, &savintr);
    sigaction(SIGQUIT, &sa, &savequit);
    sigaddset(&sa.sa_mask, SIGCHLD);
    pthread_sigmask(SIG_BLOCK, &sa.sa_mask, &saveblock);
    if ((pid = fork()) == 0) {
        sigaction(SIGINT, &savintr, (struct sigaction *)0);
        sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
        sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
        execl("/bin/sh", "sh", "-c", "--", cmd, (char *)0);
        _exit(127);
    }
    if (pid == -1) {
        stat = -1; /* errno comes from fork() */
    } else {
        while (waitpid(pid, &stat, 0) == -1) {
            if (errno != EINTR){
                stat = -1;
                break;
            }
        }
    }
    sigaction(SIGINT, &savintr, (struct sigaction *)0);
    sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
    sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
    return(stat);
}
```

Note that, while a particular implementation of *system*( ) (such as the one above) can assume a particular path for the shell, such a path is not necessarily valid on another system. The above example is not portable, and is not intended to be.

Earlier versions of this standard did not require *system*( ) to be thread-safe because it alters the process-wide disposition of the SIGINT and SIGQUIT signals. It is now required to be thread-safe to align with the ISO C standard, which (since the introduction of threads in 2011) requires that it avoids data races. However, the function is not required to coordinate the saving and restoring of the dispositions of the SIGINT and SIGQUIT signals across a set of overlapping calls, and the above example does not do so. The example also does not terminate and wait for the child process if the calling thread is cancelled, and so would leak a process ID in that event.

One reviewer suggested that an implementation of *system*( ) might want to use an environment variable such as *SHELL* to determine which command interpreter to use. The supposed implementation would use the default command interpreter if the one specified by the environment variable was not available. This would allow a user, when using an application that prompts for command lines to be processed using *system*( ), to specify a different command interpreter. Such an implementation is discouraged. If the alternate command interpreter did not follow the command line syntax specified in the Shell and Utilities volume of POSIX.1-2024, then changing *SHELL* would render *system*( ) non-conforming. This would affect applications that expected the specified behavior from *system*( ), and since the Shell and Utilities volume of POSIX.1-2024 does not mention that *SHELL* affects *system*( ), the application would not know that it needed to unset *SHELL*.

Earlier versions of this standard required the *command* string to be passed as the next argument after `"-c"` (omitting `"--"`). This meant that portable applications needed to take care not to pass a command string beginning with <hyphen-minus> (`'-'`) or <plus-sign> (`'+'`), as it would then be interpreted as containing options. Now that implementations are required to pass the `"--"`, applications no longer need to do this.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.9.5.2 (on page 543), *exec*, *pipe*( ), *pthread_atfork*( ), *wait*( )

XBD **<limits.h>**, **<signal.h>**, **<stdlib.h>**, **<sys/wait.h>**

XCU *sh*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

Issue 7

Austin Group Interpretation 1003.1-2001 #055 is applied, clarifying the thread-safety of this function and treatment of *at_fork*( ) handlers.

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XSH-ERN-30 is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0365 [627] is applied.

Issue 8

Austin Group Defect 768 is applied, adding OFD-owned file locks.

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018 standard.

Austin Group Defect 1317 is applied, making it implementation-defined whether the handlers registered with *pthread_atfork*( ) are called.

Austin Group Defect 1318 is applied, adding FD_CLOFORK.

Austin Group Defect 1440 is applied, adding a `"--"` argument to the specified *execl*( ) call.

72301 **NAME**

72302 tan, tanf, tanl — tangent function

72303 **SYNOPSIS**

72304
```
#include <math.h>
```

72305
```
double tan(double x);
```
72306
```
float tanf(float x);
```
72307
```
long double tanl(long double x);
```

72308 **DESCRIPTION**

72309 CX The functionality described on this reference page is aligned with the ISO C standard. Any
72310 conflict between the requirements described here and the ISO C standard is unintentional. This
72311 volume of POSIX.1-2024 defers to the ISO C standard.

72312 These functions shall compute the tangent of their argument *x*, measured in radians.

72313 An application wishing to check for error situations should set *errno* to zero and call
72314 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
72315 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
72316 zero, an error has occurred.

72317 **RETURN VALUE**

72318 Upon successful completion, these functions shall return the tangent of *x*.

72319 MXX If the correct value would cause underflow, and is not representable, a range error may occur,
72320 MXX and *tan*( ), *tanf*( ), and *tanl*( ) shall return 0.0, or (if IEC 60559 Floating-Point is not supported) an
72321 implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and
72322 LDBL_MIN, respectively.

72323 MX If *x* is NaN, a NaN shall be returned.

72324 If *x* is ±0, *x* shall be returned.

72325 If *x* is ±Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-
72326 defined value shall be returned.

72327 MXX If *x* is subnormal, *x* should be returned.
72328 MX If *x* is subnormal, a range error may occur and, if *x* is not returned, *tan*( ), *tanf*( ), and *tanl*( ) shall
72329 return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN,
72330 and LDBL_MIN, respectively.

72331 MXX If the correct value would cause underflow, and is representable, a range error may occur and
72332 the correct value shall be returned.

72333 XSI If the correct value would cause overflow, a range error shall occur and *tan*( ), *tanf*( ), and *tanl*( )
72334 shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL, respectively, with the same sign
72335 as the correct value of the function.

72336 **ERRORS**

72337 These functions shall fail if:

72338 MX Domain Error The value of *x* is ±Inf.

72339 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
72340 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
72341 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
72342 shall be raised.

XSI Range Error — The result overflows

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

MX Range Error — The result underflows, or the value of *x* is subnormal.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

Taking the Tangent of a 45-Degree Angle

```
#include <math.h>
...
double radians = 45.0 * M_PI / 180;
double result;
...
result = tan (radians);
```

APPLICATION USAGE

There are no known floating-point representations such that for a normal argument, *tan*(*x*) is either overflow or underflow.

These functions may lose accuracy when their argument is near a multiple of $\pi/2$ or is far from 0.0.

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

atan( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes in previous issues.

Issue 6

The *tanf*( ) and *tanl*( ) functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/64 is applied, correcting the last paragraph in the RETURN VALUE section.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0635 [68], XSH/TC1-2008/0636 [68], and XSH/TC1-2008/0637 [68] are applied.

Issue 8

Austin Group Defect 1382 is applied, rearranging the text describing the behavior when *x* is subnormal to avoid the need for two shading changes.

NAME

tanh, tanhf, tanhl — hyperbolic tangent functions

SYNOPSIS

```
#include <math.h>

double tanh(double x);
float tanhf(float x);
long double tanhl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute the hyperbolic tangent of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the hyperbolic tangent of *x*.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is ±Inf, ±1 shall be returned.

MXX If *x* is subnormal, *x* should be returned.

MX If *x* is subnormal, a range error may occur and, if *x* is not returned, *tanh*( ), *tanhf*( ), and *tanhl*( ) shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

ERRORS

These functions may fail if:

MX

Range Error	The value of *x* is subnormal. If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

atanh( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *tan*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *tanhf*( ) and *tanhl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0638 [68] is applied.

Issue 8

Austin Group Defect 1382 is applied, rearranging the text describing the behavior when *x* is subnormal to avoid the need for two shading changes.

72455 **NAME**

72456 tanl — tangent function

72457 **SYNOPSIS**

72458 `#include <math.h>`

72459 `long double tanl(long double x);`

72460 **DESCRIPTION**

72461 Refer to *tan( )*.

72462 **NAME**

72463 tcdrain — wait for transmission of output

72464 **SYNOPSIS**

72465 ```
#include <termios.h>
```

72466 ```
int tcdrain(int fildes);
```

72467 **DESCRIPTION**

72468 The *tcdrain*( ) function shall block until all output written to the object referred to by *fildes* is
72469 transmitted. The *fildes* argument is an open file descriptor associated with a terminal.

72470 Any attempts to use *tcdrain*( ) from a process which is a member of a background process group
72471 on a *fildes* associated with its controlling terminal, shall cause the process group to be sent a
72472 SIGTTOU signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring
72473 SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

72474 **RETURN VALUE**

72475 Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to
72476 indicate the error.

72477 **ERRORS**

72478 The *tcdrain*( ) function shall fail if:

72479 [EBADF] The *fildes* argument is not a valid file descriptor.

72480 [EINTR] A signal interrupted *tcdrain*( ).

72481 [EIO] The process group of the writing process is orphaned, the calling thread is not
72482 blocking SIGTTOU, and the process is not ignoring SIGTTOU.

72483 [ENOTTY] The file associated with *fildes* is not a terminal.

72484 **EXAMPLES**

72485 None.

72486 **APPLICATION USAGE**

72487 None.

72488 **RATIONALE**

72489 None.

72490 **FUTURE DIRECTIONS**

72491 None.

72492 **SEE ALSO**

72493 *tcflush*( )

72494 XBD Chapter 11 (on page 199), **<termios.h>**

72495 **CHANGE HISTORY**

72496 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

72497 **Issue 6**

72498 The following new requirements on POSIX implementations derive from alignment with the
72499 Single UNIX Specification:

72500 • In the DESCRIPTION, the final paragraph is no longer conditional on
72501 _POSIX_JOB_CONTROL. This is a FIPS requirement.

72502 • The [EIO] error is added.

72503 **Issue 7**

72504 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0639 [79], XSH/TC1-2008/0640 [79],
72505 and XSH/TC1-2008/0641 [79] are applied.

72506 **NAME**

72507 tcflow — suspend or restart the transmission or reception of data

72508 **SYNOPSIS**

72509 ```
#include <termios.h>
```

72510 ```
int tcflow(int fildes, int action);
```

72511 **DESCRIPTION**

72512 The *tcflow*( ) function shall suspend or restart transmission or reception of data on the object
72513 referred to by *fildes*, depending on the value of *action*. The *fildes* argument is an open file
72514 descriptor associated with a terminal.

72515 • If *action* is TCOOFF, output shall be suspended.

72516 • If *action* is TCOON, suspended output shall be restarted.

72517 • If *action* is TCIOFF and *fildes* refers to a terminal device, the system shall transmit a STOP
72518 character, which is intended to cause the terminal device to stop transmitting data to the
72519 system. If *fildes* is associated with a pseudo-terminal, the STOP character need not be
72520 transmitted.

72521 • If *action* is TCION and *fildes* refers to a terminal device, the system shall transmit a START
72522 character, which is intended to cause the terminal device to start transmitting data to the
72523 system. If *fildes* is associated with a pseudo-terminal, the START character need not be
72524 transmitted.

72525 The default on the opening of a terminal file is that neither its input nor its output are
72526 suspended.

72527 Attempts to use *tcflow*( ) from a process which is a member of a background process group on a
72528 *fildes* associated with its controlling terminal, shall cause the process group to be sent a
72529 SIGTTOU signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring
72530 SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

72531 **RETURN VALUE**

72532 Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to
72533 indicate the error.

72534 **ERRORS**

72535 The *tcflow*( ) function shall fail if:

72536 [EBADF] The *fildes* argument is not a valid file descriptor.

72537 [EINVAL] The *action* argument is not a supported value.

72538 [EIO] The process group of the writing process is orphaned, the calling thread is not
72539 blocking SIGTTOU, and the process is not ignoring SIGTTOU.

72540 [ENOTTY] The file associated with *fildes* is not a terminal.

72541 **EXAMPLES**

72542 None.

72543 **APPLICATION USAGE**

72544 None.

72545 **RATIONALE**

72546 None.

72547 **FUTURE DIRECTIONS**

72548 None.

72549 **SEE ALSO**

72550 *tcsendbreak( )*

72551 XBD Chapter 11 (on page 199), **<termios.h>**

72552 **CHANGE HISTORY**

72553 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

72554 **Issue 6**

72555 The following new requirements on POSIX implementations derive from alignment with the
72556 Single UNIX Specification:

72557 • The [EIO] error is added.

72558 **Issue 7**

72559 SD5-XSH-ERN-190 is applied, clarifying in the DESCRIPTION the transmission of START and
72560 STOP characters.

72561 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0642 [79], XSH/TC1-2008/0643 [79],
72562 and XSH/TC1-2008/0644 [79] are applied.

72563 **NAME**

72564 tcflush — flush non-transmitted output data, non-read input data, or both

72565 **SYNOPSIS**

72566
```
#include <termios.h>
```

72567
```
int tcflush(int fildes, int queue_selector);
```

72568 **DESCRIPTION**

72569 Upon successful completion, *tcflush*( ) shall discard data written to the object referred to by *fildes*
72570 (an open file descriptor associated with a terminal) but not transmitted, or data received but not
72571 read, depending on the value of *queue_selector*:

72572 • If *queue_selector* is TCIFLUSH, it shall flush data received but not read.

72573 • If *queue_selector* is TCOFLUSH, it shall flush data written but not transmitted.

72574 • If *queue_selector* is TCIOFLUSH, it shall flush both data received but not read and data
72575 written but not transmitted.

72576 Attempts to use *tcflush*( ) from a process which is a member of a background process group on a
72577 *fildes* associated with its controlling terminal shall cause the process group to be sent a SIGTTOU
72578 signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring SIGTTOU
72579 signals, the process shall be allowed to perform the operation, and no signal is sent.

72580 **RETURN VALUE**

72581 Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to
72582 indicate the error.

72583 **ERRORS**

72584 The *tcflush*( ) function shall fail if:

72585 [EBADF] The *fildes* argument is not a valid file descriptor.

72586 [EINVAL] The *queue_selector* argument is not a supported value.

72587 [EIO] The process group of the writing process is orphaned, the calling thread is not
72588 blocking SIGTTOU, and the process is not ignoring SIGTTOU.

72589 [ENOTTY] The file associated with *fildes* is not a terminal.

72590 **EXAMPLES**

72591 None.

72592 **APPLICATION USAGE**

72593 None.

72594 **RATIONALE**

72595 None.

72596 **FUTURE DIRECTIONS**

72597 None.

72598 **SEE ALSO**

72599 *tcdrain*( )

72600 XBD Chapter 11 (on page 199), **<termios.h>**

72601 **CHANGE HISTORY**

72602 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The Open Group Corrigendum U035/1 is applied. In the ERRORS and APPLICATION USAGE sections, references to *tcflow*( ) are replaced with *tcflush*( ).

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the final paragraph is no longer conditional on _POSIX_JOB_CONTROL. This is a FIPS requirement.
- The [EIO] error is added.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0645 [79], XSH/TC1-2008/0646 [79], and XSH/TC1-2008/0647 [79] are applied.

72614 **NAME**

72615 tcgetattr — get the parameters associated with the terminal

72616 **SYNOPSIS**

72617 ```
#include <termios.h>
```

72618 ```
int tcgetattr(int fildes, struct termios *termios_p);
```

72619 **DESCRIPTION**

72620 The *tcgetattr*( ) function shall get the parameters associated with the terminal referred to by *fildes*
72621 and store them in the **termios** structure referenced by *termios_p*. The *fildes* argument is an open
72622 file descriptor associated with a terminal.

72623 The *termios_p* argument is a pointer to a **termios** structure.

72624 The *tcgetattr*( ) operation is allowed from any process.

72625 If the terminal device supports different input and output baud rates, the baud rates stored in
72626 the **termios** structure returned by *tcgetattr*( ) shall reflect the actual baud rates, even if they are
72627 equal. If differing baud rates are not supported, the rate returned as the output baud rate shall
72628 be the actual baud rate. If the terminal device does not support split baud rates, the input baud
72629 rate stored in the **termios** structure shall be the output rate (as one of the symbolic values).

72630 **RETURN VALUE**

72631 Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to
72632 indicate the error.

72633 **ERRORS**

72634 The *tcgetattr*( ) function shall fail if:

72635 [EBADF] The *fildes* argument is not a valid file descriptor.

72636 [ENOTTY] The file associated with *fildes* is not a terminal.

72637 **EXAMPLES**

72638 None.

72639 **APPLICATION USAGE**

72640 None.

72641 **RATIONALE**

72642 Care must be taken when changing the terminal attributes. Applications should always do a
72643 *tcgetattr*( ), save the **termios** structure values returned, and then do a *tcsetattr*( ), changing only
72644 the necessary fields. The application should use the values saved from the *tcgetattr*( ) to reset the
72645 terminal state whenever it is done with the terminal. This is necessary because terminal
72646 attributes apply to the underlying port and not to each individual open instance; that is, all
72647 processes that have used the terminal see the latest attribute changes.

72648 A program that uses these functions should be written to catch all signals and take other
72649 appropriate actions to ensure that when the program terminates, whether planned or not, the
72650 terminal device's state is restored to its original state.

72651 Existing practice dealing with error returns when only part of a request can be honored is based
72652 on calls to the *ioctl*( ) function. In historical BSD and System V implementations, the
72653 corresponding *ioctl*( ) returns zero if the requested actions were semantically correct, even if
72654 some of the requested changes could not be made. Many existing applications assume this
72655 behavior and would no longer work correctly if the return value were changed from zero to –1
72656 in this case.

72657 Note that either specification has a problem. When zero is returned, it implies everything

succeeded even if some of the changes were not made. When –1 is returned, it implies everything failed even though some of the changes were made.

Applications that need all of the requested changes made to work properly should follow *tcsetattr*( ) with a call to *tcgetattr*( ) and compare the appropriate field values.

FUTURE DIRECTIONS

None.

SEE ALSO

tcsetattr( )

XBD Chapter 11 (on page 199), **<termios.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

In the DESCRIPTION, the rate returned as the input baud rate shall be the output rate. Previously, the number zero was also allowed but was obsolescent.

72672 **NAME**

72673 tcgetpgrp — get the foreground process group ID

72674 **SYNOPSIS**

72675 ```
#include <unistd.h>
```

72676 ```
pid_t tcgetpgrp(int fildes);
```

72677 **DESCRIPTION**

72678 The *tcgetpgrp*( ) function shall return the value of the process group ID of the foreground process
72679 group associated with the terminal.

72680 If there is no foreground process group, *tcgetpgrp*( ) shall return a value greater than 1 that does
72681 not match the process group ID of any existing process group.

72682 The *tcgetpgrp*( ) function is allowed from a process that is a member of a background process
72683 group; however, the information may be subsequently changed by a process that is a member of
72684 a foreground process group.

72685 **RETURN VALUE**

72686 Upon successful completion, *tcgetpgrp*( ) shall return the value of the process group ID of the
72687 foreground process associated with the terminal. Otherwise, –1 shall be returned and *errno* set to
72688 indicate the error.

72689 **ERRORS**

72690 The *tcgetpgrp*( ) function shall fail if:

72691 [EBADF] The *fildes* argument is not a valid file descriptor.

72692 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the
72693 controlling terminal.

72694 **EXAMPLES**

72695 None.

72696 **APPLICATION USAGE**

72697 None.

72698 **RATIONALE**

72699 None.

72700 **FUTURE DIRECTIONS**

72701 None.

72702 **SEE ALSO**

72703 *setsid*( ), *setpgid*( ), *tcsetpgrp*( )

72704 XBD **<sys/types.h>**, **<unistd.h>**

72705 **CHANGE HISTORY**

72706 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

72707 **Issue 6**

72708 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

72709 The following new requirements on POSIX implementations derive from alignment with the
72710 Single UNIX Specification:

72711 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
72712 required for conforming implementations of previous POSIX specifications, it was not
72713 required for UNIX applications.

72714 • In the DESCRIPTION, text previously conditional on support for _POSIX_JOB_CONTROL
72715 is now mandatory. This is a FIPS requirement.

NAME

tcgetsid — get the process group ID for the session leader for the controlling terminal

SYNOPSIS

```
#include <termios.h>

pid_t tcgetsid(int fildes);
```

DESCRIPTION

The *tcgetsid*( ) function shall obtain the process group ID of the session for which the terminal specified by *fildes* is the controlling terminal.

RETURN VALUE

Upon successful completion, *tcgetsid*( ) shall return the process group ID of the session associated with the terminal. Otherwise, a value of –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcgetsid*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

[ENOTTY] The calling process does not have a controlling terminal, or the file is not the controlling terminal.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD **<termios.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The [EACCES] error has been removed from the list of mandatory errors, and the description of [ENOTTY] has been reworded.

Issue 7

SD5-XSH-ERN-180 is applied, clarifying the RETURN VALUE section.

The *tcgetsid*( ) function is moved from the XSI option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0648 [421] is applied.

72753 **NAME**

72754 tcgetwinsize — get the size of a terminal window

72755 **SYNOPSIS**

72756
```
#include <termios.h>
```

72757
```
int tcgetwinsize(int fildes, struct winsize *winsize_p);
```

72758 **DESCRIPTION**

72759 The *tcgetwinsize*( ) function shall get the terminal window size associated with the terminal
72760 referred to by *fildes* and store it in the **winsize** structure pointed to by *winsize_p*. The *fildes*
72761 argument is an open file descriptor associated with a terminal. The *winsize_p* argument is a
72762 pointer to a **winsize** structure.

72763 If the terminal referred to by *fildes* was opened without O_TTY_INIT and is not a pseudo-
72764 terminal, and the terminal window size has not been set by a call to *tcsetwinsize*( ), the terminal
72765 window size is unspecified.

72766 If the terminal was opened with O_TTY_INIT or is a pseudo-terminal, and the terminal window
72767 size has not been set by a call to *tcsetwinsize*( ), the terminal window size shall be set to an
72768 appropriate default (see *open*( )).

72769 If the terminal window size has been set by a call to *tcsetwinsize*( ), the values set by that call
72770 shall be returned.

72771 **RETURN VALUE**

72772 Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to
72773 indicate the error.

72774 **ERRORS**

72775 The *tcgetwinsize*( ) function shall fail if:

72776 [EBADF] The *fildes* argument is not a valid file descriptor.

72777 [ENOTTY] The file associated with *fildes* is not a terminal.

72778 **EXAMPLES**

72779
```
#include <stdint.h>
```
72780
```
#include <stdio.h>
```
72781
```
#include <stdlib.h>
```
72782
```
#include <errno.h>
```
72783
```
#include <signal.h>
```
72784
```
#include <termios.h>
```
72785
```
#include <unistd.h>
```

72786
```
static volatile sig_atomic_t vrow;
```
72787
```
static volatile sig_atomic_t vcol;
```
72788
```
static volatile sig_atomic_t newsize;
```

72789
```
static void winch_handler(int signum)
```
72790
```
{
```
72791
```
    struct winsize ws;
```
72792
```
    int sav_errno = errno;
```

72793
```
    (void)signum; // prevent compiler warning that signum is unused
```

72794
```
    // set volatile vars to new winsize, or 0 if unavailable or
```



```
72795             // too large

72796             if (tcgetwinsize(STDERR_FILENO, &ws) == -1)
72797             {
72798                 vrow = vcol = 0;
72799             }
72800             else
72801             {
72802                 if (ws.ws_row <= SIG_ATOMIC_MAX && ws.ws_col <= SIG_ATOMIC_MAX)
72803                 {
72804                     vrow = ws.ws_row;
72805                     vcol = ws.ws_col;
72806                 }
72807                 else
72808                 {
72809                     vrow = vcol = 0;
72810                 }
72811             }

72812             newsize = 1;

72813             errno = sav_errno;
72814         }

72815         int main(void)
72816         {
72817             struct sigaction sa;
72818             struct winsize ws;
72819             sigset_t winch_set;
72820             char inbuf[512];

72821             sa.sa_handler = winch_handler;
72822             sigemptyset(&sa.sa_mask);
72823             sa.sa_flags = 0;
72824             sigaction(SIGWINCH, &sa, NULL);

72825             sigemptyset(&winch_set);
72826             sigaddset(&winch_set, SIGWINCH);

72827             raise(SIGWINCH); // gets the initial winsize

72828             for (;;)
72829             {
72830                 if (fgets(inbuf, sizeof inbuf, stdin) == NULL)
72831                 {
72832                     if (feof(stdin))
72833                         exit(0);
72834                     else if (errno == EINTR)
72835                         continue;
72836                     else
72837                     {
72838                         perror("Error reading stdin");
```



```
                        exit(1);
                    }
                }
                else
                {
                    if (newsize)
                    {
                        // prevent updates to volatile vars while we read them
                        sigprocmask(SIG_BLOCK, &winch_set, NULL);
                        ws.ws_row = vrow;
                        ws.ws_col = vcol;
                        sigprocmask(SIG_UNBLOCK, &winch_set, NULL);
                        newsize = 0;
                    }
                    printf("row = %3hu, col = %3hu\n", ws.ws_row, ws.ws_col);

                    // process inbuf ...
                }
            }
        }
```

APPLICATION USAGE

Applications should take care to avoid race conditions and other undefined behavior when calling *tcgetwinsize*( ) from signal handlers. A common but incorrect idiom is to establish a signal handler for SIGWINCH from which *tcgetwinsize*( ) is called to update a global **winsize** structure. This usage is incorrect as accessing a **winsize** structure is not guaranteed to be an atomic operation. Instead, applications should have *tcgetwinsize*( ) write to a local structure and copy each member the application is interested in to a global variable of type **volatile sig_atomic_t**. Furthermore, SIGWINCH should be blocked from delivery while the terminal size is read from these global variables to further avoid race conditions. A simpler alternative, if the application is structured in a suitable way, is just to set a flag in the signal handler and then call *tcgetwinsize*( ) (and clear the flag) at an appropriate place in the code if the flag has been set.

Multi-threaded applications should avoid the signal handler idiom in general. Instead, it is advised to use *sigwait*( ) to wait for the delivery of a SIGWINCH signal.

If the terminal window size changes while a process is in the background, it is not notified via SIGWINCH (which is sent only to the foreground process group). Applications can handle this case by calling *tcgetwinsize*( ) if the process receives SIGCONT, to check whether the terminal window size changed while the process was stopped.

If a background process writes to a terminal and the TOSTOP flag is clear (see XBD Section 11.2.5, on page 210), the process might not receive SIGTTOU or SIGWINCH signals and thus might not be notified when the terminal window size might have changed. Such processes must periodically poll the current terminal window size if needed.

RATIONALE

The *tcgetwinsize*( ) function is provided to allow applications to query the current terminal window size. This is necessary for applications intended to be run in terminals whose terminal window size can be changed at runtime. Conventionally, a SIGWINCH signal is delivered to a controlling terminal's foreground process group whenever its terminal window size is changed. By installing a signal handler for SIGWINCH, a process can detect the change to the controlling terminal's window size and take action, e.g. by redrawing its user interface to the new size.

72886 **FUTURE DIRECTIONS**

72887 None.

72888 **SEE ALSO**

72889 *tcsetwinsize( )*

72890 XBD **<termios.h>**

72891 **CHANGE HISTORY**

72892 First released in Issue 8.

NAME

tcsendbreak — send a break for a specific duration

SYNOPSIS

```
#include <termios.h>

int tcsendbreak(int fildes, int duration);
```

DESCRIPTION

If the terminal is using asynchronous serial data transmission, *tcsendbreak*( ) shall cause transmission of a continuous stream of zero-valued bits for a specific duration. If *duration* is 0, it shall cause transmission of zero-valued bits for at least 0.25 seconds, and not more than 0.5 seconds. If *duration* is not 0, it shall send zero-valued bits for an implementation-defined period of time.

The *fildes* argument is an open file descriptor associated with a terminal.

If the terminal is not using asynchronous serial data transmission, it is implementation-defined whether *tcsendbreak*( ) sends data to generate a break condition or returns without taking any action.

Attempts to use *tcsendbreak*( ) from a process which is a member of a background process group on a *fildes* associated with its controlling terminal shall cause the process group to be sent a SIGTTOU signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcsendbreak*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

[EIO] The process group of the writing process is orphaned, the calling thread is not blocking SIGTTOU, and the process is not ignoring SIGTTOU.

[ENOTTY] The file associated with *fildes* is not a terminal.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD Chapter 11 (on page 199), **<termios.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

72933 **Issue 6**

72934 The following new requirements on POSIX implementations derive from alignment with the
72935 Single UNIX Specification:

72936 • In the DESCRIPTION, text previously conditional on _POSIX_JOB_CONTROL is now
72937 mandated. This is a FIPS requirement.

72938 • The [EIO] error is added.

72939 **Issue 7**

72940 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0649 [79], XSH/TC1-2008/0650 [79],
72941 and XSH/TC1-2008/0651 [79] are applied.

NAME

tcsetattr — set the parameters associated with the terminal

SYNOPSIS

```
#include <termios.h>

int tcsetattr(int fildes, int optional_actions,
    const struct termios *termios_p);
```

DESCRIPTION

The *tcsetattr*( ) function shall set the parameters associated with the terminal referred to by the open file descriptor *fildes* (an open file descriptor associated with a terminal) from the **termios** structure referenced by *termios_p* as follows:

- If *optional_actions* is TCSANOW, the change shall occur immediately.
- If *optional_actions* is TCSADRAIN, the change shall occur after all output written to *fildes* is transmitted. This function should be used when changing parameters that affect output.
- If *optional_actions* is TCSAFLUSH, the change shall occur after all output written to *fildes* is transmitted, and all input so far received but not read shall be discarded before the change is made.

If the output baud rate stored in the **termios** structure pointed to by *termios_p* is the zero baud rate, B0, the modem control lines shall no longer be asserted. Normally, this shall disconnect the line.

If the input baud rate stored in the **termios** structure pointed to by *termios_p* is 0, the input baud rate given to the hardware is the same as the output baud rate stored in the **termios** structure.

The *tcsetattr*( ) function shall return successfully if it was able to perform any of the requested actions, even if some of the requested actions could not be performed. It shall set all the attributes that the implementation supports as requested and leave all the attributes not supported by the implementation unchanged. If no part of the request can be honored, it shall return –1 and set *errno* to [EINVAL]. If the input and output baud rates differ and are a combination that is not supported, neither baud rate shall be changed. A subsequent call to *tcgetattr*( ) shall return the actual state of the terminal device (reflecting both the changes made and not made in the previous *tcsetattr*( ) call). The *tcsetattr*( ) function shall not change the values found in the **termios** structure under any circumstances.

The effect of *tcsetattr*( ) is undefined if the value of the **termios** structure pointed to by *termios_p* was not derived from the result of a call to *tcgetattr*( ) on *fildes*; an application should modify only fields and flags defined by this volume of POSIX.1-2024 between the call to *tcgetattr*( ) and *tcsetattr*( ), leaving all other fields and flags unmodified.

No actions defined by this volume of POSIX.1-2024, other than a call to *tcsetattr*( ), a close of the last file descriptor in the system associated with this terminal device, or an open of the first file descriptor in the system associated with this terminal device (using the O_TTY_INIT flag if it is non-zero and the device is not a pseudo-terminal), shall cause any of the terminal attributes defined by this volume of POSIX.1-2024 to change.

If *tcsetattr*( ) is called from a process which is a member of a background process group on a *fildes* associated with its controlling terminal:

- If the calling thread is blocking SIGTTOU signals or the process is ignoring SIGTTOU signals, the operation completes normally and no signal is sent.

72985 • Otherwise, a SIGTTOU signal shall be sent to the process group.

72986 **RETURN VALUE**

72987 Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to
72988 indicate the error.

72989 **ERRORS**

72990 The *tcsetattr*( ) function shall fail if:

72991 [EBADF] The *fildes* argument is not a valid file descriptor.

72992 [EINTR] A signal interrupted *tcsetattr*( ).

72993 [EINVAL] The *optional_actions* argument is not a supported value, or an attempt was
72994 made to change an attribute represented in the **termios** structure to an
72995 unsupported value.

72996 [EIO] The process group of the writing process is orphaned, the calling thread is not
72997 blocking SIGTTOU, and the process is not ignoring SIGTTOU.

72998 [ENOTTY] The file associated with *fildes* is not a terminal.

72999 **EXAMPLES**

73000 None.

73001 **APPLICATION USAGE**

73002 If trying to change baud rates, applications should call *tcsetattr*( ) then call *tcgetattr*( ) in order to
73003 determine what baud rates were actually selected.

73004 In general, there are two reasons for an application to change the parameters associated with a
73005 terminal device:

73006 1. The device already has working parameter settings but the application needs a different
73007 behavior, such as non-canonical mode instead of canonical mode. The application sets (or
73008 clears) only a few flags or *c_cc*[] values. Typically, the terminal device in this case is either
73009 the controlling terminal for the process or a pseudo-terminal.

73010 2. The device is a modem or similar piece of equipment connected by a serial line, and it
73011 was not open before the application opened it. In this case, the application needs to
73012 initialize all of the parameter settings ``from scratch''. However, since the **termios**
73013 structure may include both standard and non-standard parameters, the application
73014 cannot just initialize the whole structure in an arbitrary way (e.g., using *memset*( )) as this
73015 may cause some of the non-standard parameters to be set incorrectly, resulting in non-
73016 conforming behavior of the terminal device. Conversely, the application cannot just set
73017 the standard parameters, assuming that the non-standard parameters will already have
73018 suitable values, as the device might previously have been used with non-conforming
73019 parameter settings (and some implementations retain the settings from one use to the
73020 next). The solution is to open the terminal device using the O_TTY_INIT flag to initialize
73021 the terminal device to have conforming parameter settings, obtain those settings using
73022 *tcgetattr*( ), and then set all of the standard parameters to the desired settings.

73023 **RATIONALE**

73024 The *tcsetattr*( ) function can be interrupted in the following situations:

73025 • It is interrupted while waiting for output to drain.

73026 • It is called from a process in a background process group and SIGTTOU is caught.

73027 See also the RATIONALE section in *tcgetattr*( ).

FUTURE DIRECTIONS

Using an input baud rate of 0 to set the input rate equal to the output rate may not necessarily be supported in a future version of this volume of POSIX.1-2024.

SEE ALSO

cfgetispeed( ), *tcgetattr*( )

XBD Chapter 11 (on page 199), **<termios.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text previously conditional on _POSIX_JOB_CONTROL is now mandated. This is a FIPS requirement.
- The [EIO] error is added.

In the DESCRIPTION, the text describing use of *tcsetattr*( ) from a process which is a member of a background process group is clarified.

Issue 7

Austin Group Interpretation 1003.1-2001 #144 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0652 [79], XSH/TC1-2008/0653 [79], and XSH/TC1-2008/0654 [79] are applied.

73048 **NAME**

73049 tcsetpgrp — set the foreground process group ID

73050 **SYNOPSIS**

73051 ```
#include <unistd.h>
```

73052 ```
int tcsetpgrp(int fildes, pid_t pgid_id);
```

73053 **DESCRIPTION**

73054 If the process has a controlling terminal, *tcsetpgrp*( ) shall set the foreground process group ID
73055 associated with the terminal to *pgid_id*. The application shall ensure that the file associated with
73056 *fildes* is the controlling terminal of the calling process and the controlling terminal is currently
73057 associated with the session of the calling process. The application shall ensure that the value of
73058 *pgid_id* matches a process group ID of a process in the same session as the calling process.

73059 Attempts to use *tcsetpgrp*( ) from a process which is a member of a background process group on
73060 a *fildes* associated with its controlling terminal shall cause the process group to be sent a
73061 SIGTTOU signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring
73062 SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

73063 **RETURN VALUE**

73064 Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to
73065 indicate the error.

73066 **ERRORS**

73067 The *tcsetpgrp*( ) function shall fail if:

73068 [EBADF] The *fildes* argument is not a valid file descriptor.

73069 [EINTR] A signal interrupted *tcsetpgrp*( ).

73070 [EINVAL] This implementation does not support the value in the *pgid_id* argument.

73071 [EIO] The process group of the writing process is orphaned, the calling thread is not
73072 blocking SIGTTOU, and the process is not ignoring SIGTTOU.

73073 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the
73074 controlling terminal, or the controlling terminal is no longer associated with
73075 the session of the calling process.

73076 [EPERM] The value of *pgid_id* is a value supported by the implementation, but does not
73077 match the process group ID of a process in the same session as the calling
73078 process.

73079 **EXAMPLES**

73080 None.

73081 **APPLICATION USAGE**

73082 None.

73083 **RATIONALE**

73084 None.

73085 **FUTURE DIRECTIONS**

73086 None.

73087 **SEE ALSO**

73088 *tcgetpgrp*( )

73089 XBD **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- In the DESCRIPTION and ERRORS sections, text previously conditional on _POSIX_JOB_CONTROL is now mandated. This is a FIPS requirement.

The normative text is updated to avoid use of the term ``must'' for application requirements.

The Open Group Corrigendum U047/4 is applied.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0655 [79] and XSH/TC1-2008/0656 [79] are applied.

Issue 8

Austin Group Defect 1126 is applied, adding the [EINTR] error.

73108 **NAME**

73109 tcsetwinsize — set the size of a terminal window

73110 **SYNOPSIS**

73111 ```
#include <termios.h>
```

73112 ```
int tcsetwinsize(int fildes, const struct winsize *winsize_p);
```

73113 **DESCRIPTION**

73114 The *tcsetwinsize*( ) function shall set the terminal window size associated with the terminal
73115 referred to by the open file descriptor *fildes* (an open file descriptor associated with a terminal)
73116 from the **winsize** structure referenced by *winsize_p*. The change shall occur immediately.

73117 If the terminal size was changed successfully, a SIGWINCH shall be delivered to the foreground
73118 process group associated with the terminal. No signal shall be delivered if the terminal size was
73119 changed to the same value it had before the *tcsetwinsize*( ) call. A SIGWINCH may also be
73120 delivered to an implementation-defined set of other processes.

73121 The *tcsetwinsize*( ) function shall return successfully if it was able to update all members of the
73122 **winsize** structure associated with the terminal.

73123 It is unspecified whether changing the terminal window size causes any changes to the size of
73124 the terminal's font.

73125 The effect of *tcsetwinsize*( ) is undefined if the value of the **winsize** structure pointed to by
73126 *winsize_p* was not derived from the result of a call to *tcgetwinsize*( ) on *fildes*; an application
73127 should modify only fields defined by this volume of POSIX.1-2024 between the call to
73128 *tcgetwinsize*( ) and *tcsetwinsize*( ), leaving all other fields unmodified.

73129 No actions defined by this volume of POSIX.1-2024, other than a call to *tcsetwinsize*( ), a close of
73130 the last file descriptor in the system associated with this terminal device, or an open of the first
73131 file descriptor in the system associated with this terminal device (using the O_TTY_INIT flag if it
73132 is non-zero and the device is not a pseudo-terminal), shall cause the terminal window size to
73133 change.

73134 If *tcsetwinsize*( ) is called from a process which is a member of a background process group on a
73135 *fildes* associated with its controlling terminal:

73136 • If the calling thread is blocking SIGTTOU signals or the process is ignoring SIGTTOU
73137 signals, the operation completes normally and no signal is sent.

73138 • Otherwise, a SIGTTOU signal shall be sent to the process group.

73139 **RETURN VALUE**

73140 Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned, the terminal
73141 window size shall not be changed, and *errno* shall be set to indicate the error.

73142 **ERRORS**

73143 The *tcsetwinsize*( ) function shall fail if:

73144 [EBADF] The *fildes* argument is not a valid file descriptor.

73145 [EIO] The process group of the writing process is orphaned, the calling thread is not
73146 blocking SIGTTOU, and the process is not ignoring SIGTTOU.

73147 [ENOTTY] The file associated with *fildes* is not a terminal.

73148 The *tcsetwinsize*( ) function may fail if:

73149 [EINVAL] An attempt was made to change an attribute represented in the **winsize**
73150 structure to an unsupported value.

73151 **EXAMPLES**

73152 None.

73153 **APPLICATION USAGE**

73154 If the terminal window of a pseudo terminal is resized, the attached manager process should
73155 invoke *tcsetwinsize*( ) to relay the new terminal window size to the foreground process group.

73156 If a process attached to the subsidiary device of a pseudo-terminal calls *tcsetwinsize*( ), the
73157 attached manager process should attempt to change the screen to reflect the new size.

73158 **RATIONALE**

73159 This standard does not mention the *ws_xpixel* and *ws_ypixel* fields that appear in the **winsize**
73160 structure of some historical implementations. With current hardware, it is not obvious that the
73161 **unsigned short** type used for these fields is sufficient and no uses of these fields in portable code
73162 were found. However, since these and other fields may be included in the **winsize** structure, the
73163 standard requires that applications use *tcgetwinsize*( ) to initialize any fields that may be
73164 provided by an implementation before setting the *ws_cols* and *ws_rows* fields using *tcsetwinsize*( )
73165 to avoid unintentionally destroying data in other fields in this structure.

73166 **FUTURE DIRECTIONS**

73167 None.

73168 **SEE ALSO**

73169 *tcgetwinsize*( )

73170 XBD **<termios.h>**

73171 **CHANGE HISTORY**

73172 First released in Issue 8.

NAME

tdelete, tfind, tsearch, twalk — manage a binary search tree

SYNOPSIS

XSI
```
#include <search.h>

void *tdelete(const void *restrict key,
    posix_tnode **restrict rootp,
    int(*compar)(const void *, const void *));
posix_tnode *tfind(const void *key,
    posix_tnode *const *rootp,
    int(*compar)(const void *, const void *));
posix_tnode *tsearch(const void *key,
    posix_tnode **rootp,
    int (*compar)(const void *, const void *));
void twalk(const posix_tnode *root,
    void (*action)(const posix_tnode *, VISIT, int));
```

DESCRIPTION

The *tdelete*( ), *tfind*( ), *tsearch*( ), and *twalk*( ) functions manipulate binary search trees. Comparisons are made with a user-supplied routine, the address of which is passed as the *compar* argument. This routine is called with two arguments, which are the pointers to the elements being compared. The application shall ensure that the user-supplied routine returns an integer less than, equal to, or greater than 0, according to whether the first argument is to be considered less than, equal to, or greater than the second argument. The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

The *tsearch*( ) function shall build and access the tree. The *key* argument is a pointer to an element to be accessed or stored. If there is a node in the tree whose element is equal to the value pointed to by *key*, a pointer to this found node shall be returned. Otherwise, the value pointed to by *key* shall be inserted (that is, a new node is created and the value of *key* is copied to this node), and a pointer to this node returned. Only pointers are copied, so the application shall ensure that the calling routine stores the data. The *rootp* argument points to a variable that points to the root node of the tree. A null pointer value for the variable pointed to by *rootp* denotes an empty tree; in this case, the variable shall be set to point to the node which shall be at the root of the new tree.

Like *tsearch*( ), *tfind*( ) shall search for a node in the tree, returning a pointer to it if found. However, if it is not found, *tfind*( ) shall return a null pointer. The arguments for *tfind*( ) are the same as for *tsearch*( ).

The *tdelete*( ) function shall delete a node from a binary search tree. The arguments are the same as for *tsearch*( ). The variable pointed to by *rootp* shall be set to a pointer to the new root of the tree if the root of the tree was changed. If the deleted node was the root of the tree and had no children, the variable pointed to by *rootp* shall be set to a null pointer. The *tdelete*( ) function shall return a pointer to the parent of the deleted node, or an unspecified non-null pointer if the deleted node was the root node, or a null pointer if the node is not found.

If *tsearch*( ) adds an element to a tree, or *tdelete*( ) successfully deletes an element from a tree, the concurrent use of that tree in another thread, or use of pointers produced by a previous call to *tfind*( ) or *tsearch*( ), produces undefined results.

The *twalk*( ) function shall traverse a binary search tree. The *root* argument is a pointer to the root node of the tree to be traversed. (Any node in a tree may be used as the root for a walk below

that node.) The argument *action* is the name of a routine to be invoked at each node. This routine is, in turn, called with three arguments. The first argument shall be the address of the node being visited. The structure pointed to by this argument is unspecified and shall not be modified by the application, but it shall be possible to cast a pointer-to-node into a pointer-to-pointer-to-element to access the element stored in the node. The second argument shall be a value from an enumeration data type:

```
typedef enum { preorder, postorder, endorder, leaf } VISIT;
```

(defined in **<search.h>**), depending on whether this is the first, second, or third time that the node is visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a leaf. The third argument shall be the level of the node in the tree, with the root being level 0.

If the calling function alters the pointer to the root, the result is undefined.

If the functions pointed to by *action* or *compar* (for any of these binary search functions) change the tree, the results are undefined.

These functions are thread-safe only as long as multiple threads do not access the same tree.

RETURN VALUE

If the node is found, both *tsearch*( ) and *tfind*( ) shall return a pointer to it. If not, *tfind*( ) shall return a null pointer, and *tsearch*( ) shall return a pointer to the inserted item.

A null pointer shall be returned by *tsearch*( ) if there is not enough space available to create a new node.

A null pointer shall be returned by *tdelete*( ), *tfind*( ), and *tsearch*( ) if *rootp* is a null pointer on entry.

The *tdelete*( ) function shall return a pointer to the parent of the deleted node, or an unspecified non-null pointer if the deleted node was the root node, or a null pointer if the node is not found.

The *twalk*( ) function shall not return a value.

In all cases where a pointer to a node is returned, the structure pointed to by the return value is unspecified and shall not be modified by the application, but it shall be possible to cast a pointer-to-node into a pointer-to-pointer-to-element to access the element stored in the node.

ERRORS

No errors are defined.

EXAMPLES

The following code reads in strings and stores structures containing a pointer to each string and a count of its length. It then walks the tree, printing out the stored strings and their lengths in alphabetical order.

```
#include <limits.h>
#include <search.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

struct element {      /* Pointers to these are stored in the tree. */
    int     count;
    char    string[];
};

posix_tnode  *root = NULL;          /* This points to the root. */
```



```
73263   int main(void)
73264   {
73265       char   str[_POSIX2_LINE_MAX+1];
73266       int    length = 0;
73267       struct element *elementptr;
73268       posix_tnode *node;
73269       void   print_node(const posix_tnode *, VISIT, int);
73270       int    node_compare(const void *, const void *);
73271       while (fgets(str, sizeof(str), stdin))  {
73272           /* Set element. */
73273           length = strlen(str);
73274           if (str[length-1] == '\n')
73275               str[--length] = '\0';
73276           elementptr = malloc(sizeof(struct element) + length + 1);
73277           strcpy(elementptr->string, str);
73278           elementptr->count = 1;
73279           /* Put element into the tree. */
73280           node = tsearch((void *)elementptr, &root, node_compare);
73281           if (node == NULL) {
73282               fprintf(stderr,
73283                       "tsearch: Not enough space available\n");
73284               exit(EXIT_FAILURE);
73285           }
73286           else if (*(struct element **)node != elementptr) {
73287               /* A node containing the element already exists */
73288               (*(struct element **)node)->count++;
73289               free(elementptr);
73290           }
73291       }
73292       twalk(root, print_node);
73293       /* Delete all nodes in the tree */
73294       while (root != NULL) {
73295           elementptr = *(struct element **)root;
73296           printf("deleting node: string = %s,  count = %d\n",
73297                  elementptr->string,
73298                  elementptr->count);
73299           tdelete((void *)elementptr, &root, node_compare);
73300           free(elementptr);
73301       }
73302       return 0;
73303   }
73304   /*
73305    *  This routine compares two nodes, based on an
73306    *  alphabetical ordering of the string field.
73307    */
73308   int
73309   node_compare(const void *node1, const void *node2)
73310   {
73311       return strcmp(((const struct element *) node1)->string,
73312           ((const struct element *) node2)->string);
```



```
}

/*
 *  This routine prints out a node, the second time
 *  twalk encounters it or if it is a leaf.
 */
void
print_node(const posix_tnode *ptr, VISIT order, int level)
{
    const struct element *p = *(const struct element **) ptr;

    if (order == postorder || order == leaf)  {
        (void) printf("string = %s,  count = %d\n",
            p->string, p->count);
    }
}
```

APPLICATION USAGE

The *root* argument to *twalk*( ) is one level of indirection less than the *rootp* arguments to *tdelete*( ) and *tsearch*( ).

There are two nomenclatures used to refer to the order in which tree nodes are visited. The *twalk*( ) function uses **preorder**, **postorder**, and **endorder** to refer respectively to visiting a node before any of its children, after its left child and before its right, and after both its children. The alternative nomenclature uses **preorder**, **inorder**, and **postorder** to refer to the same visits, which could result in some confusion over the meaning of **postorder**.

Since the return value of *tdelete*( ) is an unspecified non-null pointer in the case that the root of the tree has been deleted, applications should only use the return value of *tdelete*( ) as indication of success or failure in this case and should not assume it can be dereferenced. However, the only way that applications can tell if this case may have occurred is by checking whether the variable pointed to by *rootp* changed. Since this variable can change for other reasons (for example, tree balancing), using the return value of *tdelete*( ) as anything other than a boolean indicator is unreliable at best and is discouraged. Some implementations in this case will return a pointer to the new root of the tree (or to an empty tree if the deleted root node was the only node in the tree); other implementations return arbitrary non-null pointers.

RATIONALE

Implementations are encouraged to use balanced trees to reduce the depth of the trees that are created and improve tree search times.

FUTURE DIRECTIONS

None.

SEE ALSO

hcreate( ), *lsearch*( )

XBD **<search.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in previous issues.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

The **restrict** keyword is added to the *tdelete*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #149 is applied, clarifying concurrent use of the tree in another thread.

Austin Group Interpretation 1003.1-2001 #151 is applied, clarifying behavior for *tdelete*( ) when the deleted node is the root node.

Austin Group Interpretation 1003.1-2001 #153 is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0366 [551] is applied.

Issue 8

Austin Group Defect 1011 is applied, changing some prototypes to use **posix_tnode** instead of **void**, and changing the required behavior for *tdelete*( ) when the root of the tree changes.

Austin Group Defect 1470 is applied, changing the EXAMPLES section.

73372 **NAME**

73373 telldir — current location of a named directory stream

73374 **SYNOPSIS**

73375 XSI
```
#include <dirent.h>

long telldir(DIR *dirp);
```

73376

73377 **DESCRIPTION**

73378 The *telldir*( ) function shall obtain the current location associated with the directory stream
73379 specified by *dirp*.

73380 If the most recent operation on the directory stream was a *seekdir*( ), the directory position
73381 returned from the *telldir*( ) shall be the same as that supplied as a *loc* argument for *seekdir*( ).

73382 **RETURN VALUE**

73383 Upon successful completion, *telldir*( ) shall return the current location of the specified directory
73384 stream.

73385 **ERRORS**

73386 No errors are defined.

73387 **EXAMPLES**

73388 None.

73389 **APPLICATION USAGE**

73390 None.

73391 **RATIONALE**

73392 None.

73393 **FUTURE DIRECTIONS**

73394 None.

73395 **SEE ALSO**

73396 *fdopendir*( ), *readdir*( ), *seekdir*( )

73397 XBD **<dirent.h>**

73398 **CHANGE HISTORY**

73399 First released in Issue 2.

73400 **NAME**

73401 textdomain — text domain manipulation function

73402 **SYNOPSIS**

73403 `#include <libintl.h>`

73404 `char *textdomain(const char *`*`domainname`*`);`

73405 **DESCRIPTION**

73406 Refer to *bindtextdomain( )*.

NAME

tfind — search binary search tree

SYNOPSIS

XSI

```
#include <search.h>

void *tfind(const void *key, void *const *rootp,
    int (*compar)(const void *, const void *));
```

DESCRIPTION

Refer to *tdelete( )*.

NAME

tgamma, tgammaf, tgammal — compute gamma( ) function

SYNOPSIS

```
#include <math.h>

double tgamma(double x);
float tgammaf(float x);
long double tgammal(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall compute $\Gamma(x)$ where $\Gamma(x)$ is defined as $\int_0^\infty e^{-t}t^{x-1}dt$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the gamma of x.

If x is a negative integer, either a domain error or a pole error may occur and either a NaN (if supported) or ±Inf (if supported), respectively, or an implementation-defined value shall be returned. MX On systems that support the IEC 60559 Floating-Point option, a domain error shall occur and a NaN shall be returned.

If x is ±0, *tgamma*( ), *tgammaf*( ), and *tgammal*( ) shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL, respectively. MX On systems that support the IEC 60559 Floating-Point option, a pole error shall occur; CX otherwise, a pole error may occur.

If the correct value would cause overflow, a range error shall occur and *tgamma*( ), *tgammaf*( ), and *tgammal*( ) shall return ±HUGE_VAL, ±HUGE_VALF, or ±HUGE_VALL, respectively, with the same sign as the correct value of the function.

MXX If the correct value would cause underflow, and is not representable, a range error may occur, and *tgamma*( ), *tgammaf*( ), and *tgammal*( ) shall return 0.0, or MXX (if IEC 60559 Floating-Point is not supported) an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

MXX If the correct value would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

If x is subnormal and $1/x$ is representable, $1/x$ should be returned.

MX If x is NaN, a NaN shall be returned.

If x is +Inf, x shall be returned.

If x is –Inf, a domain error shall occur, and a NaN shall be returned.

ERRORS

These functions shall fail if:

MX Domain Error The value of x is a negative integer, or x is –Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

MX Pole Error The value of x is zero.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

Range Error The value overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

Domain Error The value of x is a negative integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error The value of x is zero or a negative integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

Range Error The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

This function is named *tgamma*( ) in order to avoid conflicts with the historical *gamma*( ) and *lgamma*( ) functions.

FUTURE DIRECTIONS

None.

73496 **SEE ALSO**

73497 *feclearexcept( )*, *fetestexcept( )*, *lgamma( )*

73498 XBD Section 4.23 (on page 109), **<math.h>**

73499 **CHANGE HISTORY**

73500 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

73501 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/65 is applied, correcting the third
73502 paragraph in the RETURN VALUE section.

73503 **Issue 7**

73504 ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #52 (SD5-XSH-ERN-85) is applied.

73505 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0660 [68], XSH/TC1-2008/0661 [320],
73506 and XSH/TC1-2008/0662 [68] are applied.

73507 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0367 [604] and XSH/TC2-2008/0368
73508 [630] are applied.

73509 **Issue 8**

73510 Austin Group Defect 1461 is applied, changing the requirements for a negative integer argument
73511 to match the ISO C standard.

73512 **NAME**

73513 thrd_create — thread creation

73514 **SYNOPSIS**

73515 `#include <threads.h>`

73516 `int thrd_create(thrd_t *thr, thrd_start_t func, void *arg);`

73517 **DESCRIPTION**

73518 CX The functionality described on this reference page is aligned with the ISO C standard. Any
73519 conflict between the requirements described here and the ISO C standard is unintentional. This
73520 volume of POSIX.1-2024 defers to the ISO C standard.

73521 The *thrd_create*( ) function shall create a new thread executing *func*(*arg*). If the *thrd_create*( )
73522 function succeeds, it shall set the object pointed to by *thr* to the identifier of the newly created
73523 thread. (A thread's identifier might be reused for a different thread once the original thread has
73524 exited and either been detached or joined to another thread.) The completion of the *thrd_create*( )
73525 function shall synchronize with the beginning of the execution of the new thread.

73526 CX The signal state of the new thread shall be initialized as follows:

73527 • The signal mask shall be inherited from the creating thread.

73528 • The set of signals pending for the new thread shall be empty.

73529 The thread-local current locale shall not be inherited from the creating thread.

73530 The floating-point environment shall be inherited from the creating thread.

73531 XSI The alternate stack shall not be inherited from the creating thread.

73532 Returning from *func* shall have the same behavior as invoking *thrd_exit*( ) with the value
73533 returned from *func*.

73534 If *thrd_create*( ) fails, no new thread shall be created and the contents of the location referenced
73535 by *thr* are undefined.

73536 CX The *thrd_create*( ) function shall not be affected if the calling thread executes a signal handler
73537 during the call.

73538 **RETURN VALUE**

73539 The *thrd_create*( ) function shall return `thrd_success` on success, or `thrd_nomem` if no
73540 memory could be allocated for the thread requested, or `thrd_error` if the request could not be
73541 CX honored, such as if the system-imposed limit on the total number of threads in a process
73542 {PTHREAD_THREADS_MAX} would be exceeded.

73543 **ERRORS**

73544 See RETURN VALUE.

73545 **EXAMPLES**

73546 None.

73547 **APPLICATION USAGE**

73548 There is no requirement on the implementation that the ID of the created thread be available
73549 before the newly created thread starts executing. The calling thread can obtain the ID of the
73550 created thread through the *thr* argument of the *thrd_create*( ) function, and the newly created
73551 thread can obtain its ID by a call to *thrd_current*( ).

73552 **RATIONALE**

73553 The *thrd_create*( ) function is not affected by signal handlers for the reasons stated in XRAT
73554 Section B.2.3 (on page 3742).

73555 **FUTURE DIRECTIONS**

73556 None.

73557 **SEE ALSO**

73558 *pthread_create*( ), *thrd_current*( ), *thrd_detach*( ), *thrd_exit*( ), *thrd_join*( )

73559 XBD Section 4.15.2 (on page 104), **<threads.h>**

73560 **CHANGE HISTORY**

73561 First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

thrd_current — get the calling thread ID

SYNOPSIS

```
#include <threads.h>

thrd_t thrd_current(void);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *thrd_current*( ) function shall identify the thread that called it.

RETURN VALUE

The *thrd_current*( ) function shall return the thread ID of the thread that called it.

The *thrd_current*( ) function shall always be successful. No return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_self( ), *thrd_create*( ), *thrd_equal*( )

XBD **<threads.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

thrd_detach — detach a thread

SYNOPSIS

```
#include <threads.h>

int thrd_detach(thrd_t thr);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *thrd_detach*( ) function shall change the thread *thr* from joinable to detached, indicating to the implementation that any resources allocated to the thread can be reclaimed when that thread terminates. The application shall ensure that the thread identified by *thr* has not been previously detached or joined with another thread.

CX The *thrd_detach*( ) function shall not be affected if the calling thread executes a signal handler during the call.

RETURN VALUE

The *thrd_detach*( ) function shall return `thrd_success` on success or `thrd_error` if the request could not be honored.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *thrd_detach*( ) function is not affected by signal handlers for the reasons stated in XRAT Section B.2.3 (on page 3742).

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_detach( ), *thrd_create*( ), *thrd_join*( )

XBD **<threads.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

thrd_equal — compare thread IDs

SYNOPSIS

```
#include <threads.h>

int thrd_equal(thrd_t thr0, thrd_t thr1);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *thrd_equal*( ) function shall determine whether the thread identified by *thr0* refers to the thread identified by *thr1*.

CX The *thrd_equal*( ) function shall not be affected if the calling thread executes a signal handler during the call.

RETURN VALUE

The *thrd_equal*( ) function shall return a non-zero value if *thr0* and *thr1* are equal; otherwise, zero shall be returned.

CX If either *thr0* or *thr1* is not a valid thread ID and is not equal to PTHREAD_NULL (which is defined in **<pthread.h>**), the behavior is undefined.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

See the RATIONALE section for *pthread_equal*( ).

The *thrd_equal*( ) function is not affected by signal handlers for the reasons stated in XRAT Section B.2.3 (on page 3742).

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_equal( ), *thrd_current*( )

XBD **<pthread.h>**, **<threads.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

thrd_exit — thread termination

SYNOPSIS

```
#include <threads.h>

_Noreturn void thrd_exit(int res);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX For every thread-specific storage key (regardless of whether it has type **tss_t** or **pthread_key_t**) which was created with a non-null destructor and for which the value is non-null, *thrd_exit*( ) shall set the value associated with the key to a null pointer value and then invoke the destructor with its previous value. The order in which destructors are invoked is unspecified.

If after this process there remain keys with both non-null destructors and values, the CX implementation shall repeat this process up to {PTHREAD_DESTRUCTOR_ITERATIONS} times.

Following this, the *thrd_exit*( ) function shall terminate execution of the calling thread and shall CX set its exit status to *res*. Thread termination shall not release any application visible process resources, including, but not limited to, mutexes and file descriptors, nor shall it perform any process-level cleanup actions, including, but not limited to, calling any *atexit*( ) routines that might exist.

An implicit call to *thrd_exit*( ) is made when a thread that was created using *thrd_create*( ) returns from the start routine that was used to create it (see *thrd_create*( )).

CX The behavior of *thrd_exit*( ) is undefined if called from a destructor function that was invoked as a result of either an implicit or explicit call to *thrd_exit*( ).

The process shall exit with an exit status of zero after the last thread has been terminated. The behavior shall be as if the implementation called *exit*( ) with a zero argument at thread termination time.

RETURN VALUE

This function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Calls to *thrd_exit*( ) should not be made from threads created using *pthread_create*( ) or via a SIGEV_THREAD notification, as their exit status has a different type (**void *** instead of **int**). If *thrd_exit*( ) is called from the initial thread and it is not the last thread to terminate, other threads should not try to obtain its exit status using *pthread_join*( ).

RATIONALE

The normal mechanism by which a thread that was started using *thrd_create*( ) terminates is to return from the function that was specified in the *thrd_create*( ) call that started it. The *thrd_exit*( ) function provides the capability for such a thread to terminate without requiring a return from the start routine of that thread, thereby providing a function analogous to *exit*( ).

Regardless of the method of thread termination, the destructors for any existing thread-specific

data are executed.

FUTURE DIRECTIONS

None.

SEE ALSO

exit( ), *pthread_create*( ), *thrd_join*( )

XBD **<threads.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

73713 **NAME**

73714 thrd_join — wait for thread termination

73715 **SYNOPSIS**

73716 `#include <threads.h>`

73717 `int thrd_join(thrd_t thr, int *res);`

73718 **DESCRIPTION**

73719 CX The functionality described on this reference page is aligned with the ISO C standard. Any
73720 conflict between the requirements described here and the ISO C standard is unintentional. This
73721 volume of POSIX.1-2024 defers to the ISO C standard.

73722 The *thrd_join*( ) function shall join the thread identified by *thr* with the current thread by
73723 blocking until the other thread has terminated. If the parameter *res* is not a null pointer,
73724 *thrd_join*( ) shall store the thread's exit status in the integer pointed to by *res*. The termination of
73725 the other thread shall synchronize with the completion of the *thrd_join*( ) function. The
73726 application shall ensure that the thread identified by *thr* has not been previously detached or
73727 joined with another thread.

73728 The results of multiple simultaneous calls to *thrd_join*( ) specifying the same target thread are
73729 undefined.

73730 The behavior is undefined if the value specified by the *thr* argument to *thrd_join*( ) refers to the
73731 calling thread.

73732 CX It is unspecified whether a zombie thread counts against {PTHREAD_THREADS_MAX}.

73733 If *thr* refers to a thread that was created using *pthread_create*( ) or via a SIGEV_THREAD
73734 notification and the thread terminates, or has already terminated, by returning from its start
73735 routine, the behavior of *thrd_join*( ) is undefined. If *thr* refers to a thread that terminates, or has
73736 already terminated, by calling *pthread_exit*( ) or by being cancelled, the behavior of *thrd_join*( ) is
73737 undefined.

73738 The *thrd_join*( ) function shall not be affected if the calling thread executes a signal handler
73739 during the call.

73740 **RETURN VALUE**

73741 The *thrd_join*( ) function shall return `thrd_success` on success or `thrd_error` if the request
73742 could not be honored.

73743 CX It is implementation-defined whether *thrd_join*( ) detects deadlock situations; if it does detect
73744 them, it shall return `thrd_error` when one is detected.

73745 **ERRORS**

73746 See RETURN VALUE.

73747 **EXAMPLES**

73748 None.

73749 **APPLICATION USAGE**

73750 None.

73751 **RATIONALE**

73752 The *thrd_join*( ) function provides a simple mechanism allowing an application to wait for a
73753 thread to terminate. After the thread terminates, the application may then choose to clean up
73754 resources that were used by the thread. For instance, after *thrd_join*( ) returns, any application-
73755 provided stack storage could be reclaimed.

73756 The *thrd_join*( ) or *thrd_detach*( ) function should eventually be called for every thread that is

created using *thrd_create*( ) so that storage associated with the thread may be reclaimed.

The *thrd_join*( ) function cannot be used to obtain the exit status of a thread that was created using *pthread_create*( ) or via a SIGEV_THREAD notification and which terminates by returning from its start routine, or of a thread that terminates by calling *pthread_exit*( ), because such threads have a **void** * exit status, instead of the **int** that *thrd_join*( ) returns via its *res* argument.

The *thrd_join*( ) function cannot be used to obtain the exit status of a thread that terminates by being cancelled because it has no way to indicate that a thread was cancelled. (The *pthread_join*( ) function does this by returning a reserved **void** * exit status; it is not possible to reserve an **int** value for this purpose without introducing a conflict with the ISO C standard.) The standard developers considered adding a `thrd_canceled` enumeration constant that *thrd_join*( ) would return in this case. However, this return would be unexpected in code that is written to conform to the ISO C standard, and it would also not solve the problem that threads which use only ISO C **<threads.h>** interfaces (such as ones created by third party libraries written to conform to the ISO C standard) have no way to handle being cancelled, as the ISO C standard does not provide cancellation cleanup handlers.

The *thrd_join*( ) function is not affected by signal handlers for the reasons stated in XRAT Section B.2.3 (on page 3742).

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_create( ), *pthread_exit*( ), *pthread_join*( ), *thrd_create*( ), *thrd_exit*( )

XBD Section 4.15.2 (on page 104), **<threads.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

thrd_sleep — suspend execution for an interval

SYNOPSIS

```
#include <threads.h>

int thrd_sleep(const struct timespec *duration,
    struct timespec *remaining);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *thrd_sleep*( ) function shall suspend execution of the calling thread until either the interval specified by *duration* has elapsed or a signal is delivered to the calling thread whose action is to invoke a signal-catching function or to terminate the process. If interrupted by a signal and the *remaining* argument is not null, the amount of time remaining (the requested interval minus the time actually slept) shall be stored in the interval it points to. The *duration* and *remaining* arguments can point to the same object.

The suspension time may be longer than requested because the interval is rounded up to an integer multiple of the sleep resolution or because of the scheduling of other activity by the system. But, except for the case of being interrupted by a signal, the suspension time shall not be less than that specified, as measured by the system clock TIME_UTC.

RETURN VALUE

The *thrd_sleep*( ) function shall return zero if the requested time has elapsed, –1 if it has been interrupted by a signal, or a negative value (which may also be –1) if it fails for any other reason.

CX If it returns a negative value, it shall set *errno* to indicate the error.

ERRORS

CX The *thrd_sleep*( ) function shall fail if:

[EINTR] The *thrd_sleep*( ) function was interrupted by a signal.

[EINVAL] The *duration* argument specified a nanosecond value less than zero or greater than or equal to 1 000 million.

EXAMPLES

None.

APPLICATION USAGE

Since the return value may be –1 for errors other than [EINTR], applications should examine *errno* to distinguish [EINTR] from other errors (and thus determine whether the unslept time is available in the interval pointed to by *remaining*).

RATIONALE

The *thrd_sleep*( ) function is identical to the *nanosleep*( ) function except that the return value may be any negative value when it fails with an error other than [EINTR].

FUTURE DIRECTIONS

None.

SEE ALSO

nanosleep( )

XBD **<threads.h>**, **<time.h>**

73824 **CHANGE HISTORY**

73825 First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

thrd_yield — yield the processor

SYNOPSIS

```
#include <threads.h>

void thrd_yield(void);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *thrd_yield*( ) function shall force the running thread to relinquish the processor until it again becomes the head of its thread list.

RETURN VALUE

This function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

See the APPLICATION USAGE section for *sched_yield*( ).

RATIONALE

The *thrd_yield*( ) function is identical to the *sched_yield*( ) function except that it does not return a value.

FUTURE DIRECTIONS

None.

SEE ALSO

sched_yield( )

XBD **<threads.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

time — get time

SYNOPSIS

```
#include <time.h>

time_t time(time_t *tloc);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *time*( ) function shall return the value of time in seconds since the Epoch.

The *tloc* argument points to an area where the return value is also stored. If *tloc* is a null pointer, no value is stored.

RETURN VALUE

Upon successful completion, *time*( ) shall return the value of time. Otherwise, (**time_t**)–1 shall be returned.

ERRORS

The *time*( ) function may fail if:

CX [EOVERFLOW] The number of seconds since the Epoch will not fit in an object of type **time_t**.

EXAMPLES

Getting the Current Time

The following example uses the *time*( ) function to calculate the time elapsed, in seconds, since the Epoch, *localtime*( ) to convert that value to a broken-down time, and *asctime*( ) to convert the broken-down time values into a printable string.

```
#include <stdio.h>
#include <time.h>

int main(void)
{
time_t result;

    result = time(NULL);
    printf("%s%ju secs since the Epoch\n",
        asctime(localtime(&result)),
            (uintmax_t)result);
    return(0);
}
```

This example writes the current time to *stdout* in a form like this:

```
Wed Jun 26 10:32:15 1996
835810335 secs since the Epoch
```


73892 **Timing an Event**

73893 The following example gets the current time, prints it out in the user's format, and prints the
73894 number of minutes to an event being timed.

```
73895 #include <time.h>
73896 #include <stdio.h>
73897 ...
73898 time_t now;
73899 int minutes_to_event;
73900 ...
73901 time(&now);
73902 minutes_to_event = ...;
73903 printf("The time is ");
73904 puts(asctime(localtime(&now)));
73905 printf("There are %d minutes to the event.\n",
73906     minutes_to_event);
73907 ...
```

73908 **APPLICATION USAGE**

73909 None.

73910 **RATIONALE**

73911 The *time*( ) function returns a value in seconds while *clock_gettime*( ) returns a **struct timespec**
73912 (seconds and nanoseconds) and is therefore capable of returning more precise times. The *times*( )
73913 function is also capable of more precision than *time*( ) as it returns a value in clock ticks,
73914 although it returns the elapsed time since an arbitrary point such as system boot time, not since
73915 the epoch.

73916 Earlier versions of this standard allowed the width of **time_t** to be less than 64 bits. A 32-bit
73917 signed integer (as used in many historical implementations) fails in the year 2038, and although
73918 a 32-bit unsigned integer does not fail until 2106 the preferred solution is to make **time_t** wider
73919 rather than to make it unsigned.

73920 On some systems the *time*( ) function is implemented using a system call that does not return an
73921 error condition in addition to the return value. On these systems it is impossible to differentiate
73922 between valid and invalid return values and hence overflow conditions cannot be reliably
73923 detected.

73924 The use of the **<time.h>** header instead of **<sys/types.h>** allows compatibility with the ISO C
73925 standard.

73926 Many historical implementations (including Version 7) and the 1984 /usr/group standard use
73927 **long** instead of **time_t**. This volume of POSIX.1-2024 uses the latter type in order to agree with
73928 the ISO C standard.

73929 **FUTURE DIRECTIONS**

73930 None.

73931 **SEE ALSO**

73932 *asctime*( ), *clock*( ), *clock_getres*( ), *ctime*( ), *difftime*( ), *futimens*( ), *gmtime*( ), *localtime*( ), *mktime*( ),
73933 *strftime*( ), *strptime*( ), *times*( )

73934 XBD **<time.h>**

73935 **CHANGE HISTORY**

73936 First released in Issue 1. Derived from Issue 1 of the SVID.

73937 **Issue 6**

73938 Extensions beyond the ISO C standard are marked.

73939 The EXAMPLES, RATIONALE, and FUTURE DIRECTIONS sections are added.

73940 **Issue 7**

73941 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0663 [106], XSH/TC1-2008/0664 [350],
73942 XSH/TC1-2008/0665 [106], XSH/TC1-2008/0666 [350], and XSH/TC1-2008/0667 [350] are
73943 applied.

73944 **Issue 8**

73945 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

73946 Austin Group Defect 1462 is applied, changing the RATIONALE and FUTURE DIRECTIONS
73947 sections.

73948 **NAME**

73949 timer_create — create a per-process timer

73950 **SYNOPSIS**

73951 CX
```
#include <signal.h>
#include <time.h>

int timer_create(clockid_t clockid, struct sigevent *restrict evp,
    timer_t *restrict timerid);
```

73955 **DESCRIPTION**

73956 The *timer_create*( ) function shall create a per-process timer using the specified clock, *clock_id*, as
73957 the timing base. The *timer_create*( ) function shall return, in the location referenced by *timerid*, a
73958 timer ID of type **timer_t** used to identify the timer in timer requests. This timer ID shall be
73959 unique within the calling process until the timer is deleted. The particular clock, *clock_id*, is
73960 defined in **<time.h>**. The timer whose ID is returned shall be in a disarmed state upon return
73961 from *timer_create*( ).

73962 The *evp* argument, if non-NULL, points to a **sigevent** structure. This structure, allocated by the
73963 application, defines the asynchronous notification to occur as specified in Section 2.4.1 (on page
73964 513) when the timer expires. If the *evp* argument is NULL, the effect is as if the *evp* argument
73965 pointed to a **sigevent** structure with the *sigev_notify* member having the value SIGEV_SIGNAL,
73966 the *sigev_signo* having a default signal number, and the *sigev_value* member having the value of
73967 the timer ID.

73968 Each implementation shall define a set of clocks that can be used as timing bases for per-process
73969 timers. All implementations shall support CLOCK_REALTIME and CLOCK_MONOTONIC as
73970 values for *clock_id*.

73971 Per-process timers shall not be inherited by a child process across a *fork*( ) and shall be disarmed
73972 and deleted by an *exec*.

73973 CPT If _POSIX_CPUTIME is defined, implementations shall support *clock_id* values representing the
73974 CPU-time clock of the calling process.

73975 TCT If _POSIX_THREAD_CPUTIME is defined, implementations shall support *clock_id* values
73976 representing the CPU-time clock of the calling thread.

73977 CPT|TCT It is implementation-defined whether a *timer_create*( ) function will succeed if the value defined
73978 by *clock_id* corresponds to the CPU-time clock of a process or thread different from the process
73979 or thread invoking the function.

73980 TSA If *evp*–>*sigev_sigev_notify* is SIGEV_THREAD and *sev*–>*sigev_notify_attributes* is not NULL, if the
73981 attribute pointed to by *sev*–>*sigev_notify_attributes* has a thread stack address specified by a call
73982 to *pthread_attr_setstack*( ), the results are unspecified if the signal is generated more than once.

73983 **RETURN VALUE**

73984 If the call succeeds, *timer_create*( ) shall return zero and update the location referenced by *timerid*
73985 to a **timer_t**, which can be passed to the per-process timer calls. If an error occurs, the function
73986 shall return a value of –1 and set *errno* to indicate the error. The value of *timerid* is undefined if
73987 an error occurs.

73988 **ERRORS**

73989 The *timer_create*( ) function shall fail if:

73990 [EAGAIN] The system lacks sufficient signal queuing resources to honor the request.

[EAGAIN] The calling process has already created all of the timers it is allowed by this implementation.

[EINVAL] The specified clock ID is not defined.

CPT|TCT [ENOTSUP] The implementation does not support the creation of a timer attached to the CPU-time clock that is specified by *clock_id* and associated with a process or thread different from the process or thread invoking *timer_create*( ).

EXAMPLES

None.

APPLICATION USAGE

If a timer is created which has *evp*−>*sigev_sigev_notify* set to SIGEV_THREAD and the attribute pointed to by *evp*−>*sigev_notify_attributes* has a thread stack address specified by a call to *pthread_attr_setstack*( ), the memory dedicated as a thread stack cannot be recovered. The reason for this is that the threads created in response to a timer expiration are created detached, or in an unspecified way if the thread attribute's *detachstate* is PTHREAD_CREATE_JOINABLE. In neither case is it valid to call *pthread_join*( ), which makes it impossible to determine the lifetime of the created thread which thus means the stack memory cannot be reused.

RATIONALE

Periodic Timer Overrun and Resource Allocation

The specified timer facilities may deliver realtime signals (that is, queued signals) on implementations that support this option. Since realtime applications cannot afford to lose notifications of asynchronous events, like timer expirations or asynchronous I/O completions, it must be possible to ensure that sufficient resources exist to deliver the signal when the event occurs. In general, this is not a difficulty because there is a one-to-one correspondence between a request and a subsequent signal generation. If the request cannot allocate the signal delivery resources, it can fail the call with an [EAGAIN] error.

Periodic timers are a special case. A single request can generate an unspecified number of signals. This is not a problem if the requesting process can service the signals as fast as they are generated, thus making the signal delivery resources available for delivery of subsequent periodic timer expiration signals. But, in general, this cannot be assured—processing of periodic timer signals may ``overrun''; that is, subsequent periodic timer expirations may occur before the currently pending signal has been delivered.

Also, for signals, according to the POSIX.1-1990 standard, if subsequent occurrences of a pending signal are generated, it is implementation-defined whether a signal is delivered for each occurrence. This is not adequate for some realtime applications. So a mechanism is required to allow applications to detect how many timer expirations were delayed without requiring an indefinite amount of system resources to store the delayed expirations.

The specified facilities provide for an overrun count. The overrun count is defined as the number of extra timer expirations that occurred between the time a timer expiration signal is generated and the time the signal is delivered. The signal-catching function, if it is concerned with overruns, can retrieve this count on entry. With this method, a periodic timer only needs one ``signal queuing resource'' that can be allocated at the time of the *timer_create*( ) function call.

A function is defined to retrieve the overrun count so that an application need not allocate static storage to contain the count, and an implementation need not update this storage asynchronously on timer expirations. But, for some high-frequency periodic applications, the overhead of an additional system call on each timer expiration may be prohibitive. The functions, as defined, permit an implementation to maintain the overrun count in user space,

74037 associated with the *timerid*. The *timer_getoverrun*( ) function can then be implemented as a macro
74038 that uses the *timerid* argument (which may just be a pointer to a user space structure containing
74039 the counter) to locate the overrun count with no system call overhead. Other implementations,
74040 less concerned with this class of applications, can avoid the asynchronous update of user space
74041 by maintaining the count in a system structure at the cost of the extra system call to obtain it.

74042 **Timer Expiration Signal Parameters**

74043 The realtime signals functionality supports an application-specific datum that is delivered to the
74044 extended signal handler. This value is explicitly specified by the application, along with the
74045 signal number to be delivered, in a **sigevent** structure. The type of the application-defined value
74046 can be either an integer constant or a pointer. This explicit specification of the value, as opposed
74047 to always sending the timer ID, was selected based on existing practice.

74048 It is common practice for realtime applications (on non-POSIX systems or realtime extended
74049 POSIX systems) to use the parameters of event handlers as the case label of a switch statement or
74050 as a pointer to an application-defined data structure. Since *timer_id*s are dynamically allocated
74051 by the *timer_create*( ) function, they can be used for neither of these functions without additional
74052 application overhead in the signal handler; for example, to search an array of saved timer IDs to
74053 associate the ID with a constant or application data structure.

74054 **FUTURE DIRECTIONS**

74055 None.

74056 **SEE ALSO**

74057 *clock_getres*( ), *timer_delete*( ), *timer_getoverrun*( )

74058 XBD **<signal.h>**, **<time.h>**

74059 **CHANGE HISTORY**

74060 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

74061 **Issue 6**

74062 The *timer_create*( ) function is marked as part of the Timers option.

74063 The [ENOSYS] error condition has been removed as stubs need not be provided if an
74064 implementation does not support the Timers option.

74065 CPU-time clocks are added for alignment with IEEE Std 1003.1d-1999.

74066 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding the
74067 requirement for the CLOCK_MONOTONIC clock under the Monotonic Clock option.

74068 The **restrict** keyword is added to the *timer_create*( ) prototype for alignment with the
74069 ISO/IEC 9899:1999 standard.

74070 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/138 is applied, updating the
74071 DESCRIPTION and APPLICATION USAGE sections to describe the case when a timer is created
74072 with the notification method set to SIGEV_THREAD.

74073 **Issue 7**

74074 The *timer_create*( ) function is moved from the Timers option to the Base.

74075 **Issue 8**

74076 Austin Group Defect 1116 is applied, removing a reference to the Realtime Signals Extension
74077 option that existed in earlier versions of this standard.

74078 Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.

NAME

timer_delete — delete a per-process timer

SYNOPSIS

CX
```
#include <time.h>

int timer_delete(timer_t timerid);
```

DESCRIPTION

The *timer_delete*( ) function deletes the specified timer, *timerid*, previously created by the *timer_create*( ) function. If the timer is armed when *timer_delete*( ) is called, the behavior shall be as if the timer is automatically disarmed before removal. The disposition of pending signals for the deleted timer is unspecified.

The behavior is undefined if the value specified by the *timerid* argument to *timer_delete*( ) does not correspond to a timer ID returned by *timer_create*( ) but not yet deleted by *timer_delete*( ).

RETURN VALUE

If successful, the *timer_delete*( ) function shall return a value of zero. Otherwise, the function shall return a value of –1 and set *errno* to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *timerid* argument to *timer_delete*( ) does not correspond to a timer ID returned by *timer_create*( ) but not yet deleted by *timer_delete*( ), it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

timer_create( )

XBD **<time.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *timer_delete*( ) function is marked as part of the Timers option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Timers option.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/139 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

Issue 7

The *timer_delete*( ) function is moved from the Timers option to the Base.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0369 [659] is applied.

NAME

timer_getoverrun, timer_gettime, timer_settime — per-process timers

SYNOPSIS

CX
```
#include <time.h>

int timer_getoverrun(timer_t timerid);
int timer_gettime(timer_t timerid, struct itimerspec *value);
int timer_settime(timer_t timerid, int flags,
    const struct itimerspec *restrict value,
    struct itimerspec *restrict ovalue);
```

DESCRIPTION

The *timer_gettime*( ) function shall store the amount of time until the specified timer, *timerid*, expires and the reload value of the timer into the space pointed to by the *value* argument. The *it_value* member of this structure shall contain the amount of time before the timer expires, or zero if the timer is disarmed. This value is returned as the interval until timer expiration, even if the timer was armed with absolute time. The *it_interval* member of *value* shall contain the reload value last set by *timer_settime*( ).

The *timer_settime*( ) function shall set the time until the next expiration of the timer specified by *timerid* from the *it_value* member of the *value* argument and arm the timer if the *it_value* member of *value* is non-zero. If the specified timer was already armed when *timer_settime*( ) is called, this call shall reset the time until next expiration to the *value* specified. If the *it_value* member of *value* is zero, the timer shall be disarmed. The effect of disarming or resetting a timer with pending expiration notifications is unspecified.

If the flag TIMER_ABSTIME is not set in the argument *flags*, *timer_settime*( ) shall behave as if the time until next expiration is set to be equal to the interval specified by the *it_value* member of *value*. That is, the timer shall expire in *it_value* nanoseconds from when the call is made. If the flag TIMER_ABSTIME is set in the argument *flags*, *timer_settime*( ) shall behave as if the time until next expiration is set to be equal to the difference between the absolute time specified by the *it_value* member of *value* and the current value of the clock associated with *timerid*. That is, the timer shall expire when the clock reaches the value specified by the *it_value* member of *value*. If the specified time has already passed, the function shall succeed and the expiration notification shall be made.

The reload value of the timer shall be set to the value specified by the *it_interval* member of *value*. When a timer is armed with a non-zero *it_interval*, a periodic (or repetitive) timer is specified.

Time values that are between two consecutive non-negative integer multiples of the resolution of the specified timer shall be rounded up to the larger multiple of the resolution. Quantization error shall not cause the timer to expire earlier than the rounded time value.

If the argument *ovalue* is not NULL, the *timer_settime*( ) function shall store, in the location referenced by *ovalue*, a value representing the previous amount of time before the timer would have expired, or zero if the timer was disarmed, together with the previous timer reload value. Timers shall not expire before their scheduled time.

Only a single signal shall be queued to the process for a given timer at any point in time. When a timer for which a signal is still pending expires, no signal shall be queued, and a timer overrun shall occur. When a timer expiration signal is delivered to or accepted by a process, the *timer_getoverrun*( ) function shall return the timer expiration overrun count for the specified timer. The overrun count returned contains the number of extra timer expirations that occurred between the time the signal was generated (queued) and when it was delivered or accepted, up

to but not including an implementation-defined maximum of {DELAYTIMER_MAX}. If the number of such extra expirations is greater than or equal to {DELAYTIMER_MAX}, then the overrun count shall be set to {DELAYTIMER_MAX}. The value returned by *timer_getoverrun*( ) shall apply to the most recent expiration signal delivery or acceptance for the timer. If no expiration signal has been delivered for the timer, the return value of *timer_getoverrun*( ) is unspecified.

The behavior is undefined if the value specified by the *timerid* argument to *timer_getoverrun*( ), *timer_gettime*( ), or *timer_settime*( ) does not correspond to a timer ID returned by *timer_create*( ) but not yet deleted by *timer_delete*( ).

RETURN VALUE

If the *timer_getoverrun*( ) function succeeds, it shall return the timer expiration overrun count as explained above.

If the *timer_gettime*( ) or *timer_settime*( ) functions succeed, a value of 0 shall be returned.

If an error occurs for any of these functions, the value –1 shall be returned, and *errno* set to indicate the error.

ERRORS

The *timer_settime*( ) function shall fail if:

[EINVAL] A *value* structure specified a nanosecond value less than zero or greater than or equal to 1 000 million, and the *it_value* member of that structure did not specify zero seconds and nanoseconds.

The *timer_settime*( ) function may fail if:

[EINVAL] The *it_interval* member of *value* is not zero and the timer was created with notification by creation of a new thread (*sigev_sigev_notify* was SIGEV_THREAD) and a fixed stack address has been set in the thread attribute pointed to by *sigev_notify_attributes*.

EXAMPLES

None.

APPLICATION USAGE

Using fixed stack addresses is problematic when timer expiration is signaled by the creation of a new thread. Since it cannot be assumed that the thread created for one expiration is finished before the next expiration of the timer, it could happen that two threads use the same memory as a stack at the same time. This is invalid and produces undefined results.

RATIONALE

Practical clocks tick at a finite rate, with rates of 100 hertz and 1 000 hertz being common. The inverse of this tick rate is the clock resolution, also called the clock granularity, which in either case is expressed as a time duration, being 10 milliseconds and 1 millisecond respectively for these common rates. The granularity of practical clocks implies that if one reads a given clock twice in rapid succession, one may get the same time value twice; and that timers must wait for the next clock tick after the theoretical expiration time, to ensure that a timer never returns too soon. Note also that the granularity of the clock may be significantly coarser than the resolution of the data format used to set and get time and interval values. Also note that some implementations may choose to adjust time and/or interval values to exactly match the ticks of the underlying clock.

This volume of POSIX.1-2024 defines functions that allow an application to determine the implementation-supported resolution for the clocks and requires an implementation to document the resolution supported for timers and *nanosleep*( ) if they differ from the supported

clock resolution. This is more of a procurement issue than a runtime application issue.

If an implementation detects that the value specified by the *timerid* argument to *timer_getoverrun*( ), *timer_gettime*( ), or *timer_settime*( ) does not correspond to a timer ID returned by *timer_create*( ) but not yet deleted by *timer_delete*( ), it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getres( ), *timer_create*( )

XBD **<time.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *timer_getoverrun*( ), *timer_gettime*( ), and *timer_settime*( ) functions are marked as part of the Timers option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Timers option.

The [EINVAL] error condition is updated to include the following: ``and the *it_value* member of that structure did not specify zero seconds and nanoseconds.'' This change is for IEEE PASC Interpretation 1003.1 #89.

The DESCRIPTION for *timer_getoverrun*( ) is updated to clarify that ``If no expiration signal has been delivered for the timer, or if the Realtime Signals Extension is not supported, the return value of *timer_getoverrun*( ) is unspecified''.

The **restrict** keyword is added to the *timer_settime*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/140 is applied, updating the ERRORS section so that the mandatory [EINVAL] error (``The *timerid* argument does not correspond to an ID returned by *timer_create*( ) but not yet deleted by *timer_delete*( )'') becomes optional.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/141 is applied, updating the ERRORS section to include an optional [EINVAL] error for the case when a timer is created with the notification method set to SIGEV_THREAD. APPLICATION USAGE text is also added.

Issue 7

The *timer_getoverrun*( ), *timer_gettime*( ), and *timer_settime*( ) functions are moved from the Timers option to the Base.

Functionality relating to the Realtime Signals Extension option is moved to the Base.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0370 [659] is applied.

NAME

times — get process and waited-for child process times

SYNOPSIS

```
#include <sys/times.h>

clock_t times(struct tms *buffer);
```

DESCRIPTION

The *times*( ) function shall fill the **tms** structure pointed to by *buffer* with time-accounting information. The **tms** structure is defined in **<sys/times.h>**.

All times are measured in terms of the number of clock ticks used.

The times of a terminated child process shall be included in the *tms_cutime* and *tms_cstime* elements of the parent when *wait*( ), *waitid*( ), or *waitpid*( ) returns the process ID of this terminated child. If a child process has not waited for its children, their times shall not be included in its times.

- The *tms_utime* structure member is the CPU time charged for the execution of user instructions of the calling process.
- The *tms_stime* structure member is the CPU time charged for execution by the system on behalf of the calling process.
- The *tms_cutime* structure member is the sum of the *tms_utime* and *tms_cutime* times of the child processes.
- The *tms_cstime* structure member is the sum of the *tms_stime* and *tms_cstime* times of the child processes.

RETURN VALUE

Upon successful completion, *times*( ) shall return the elapsed real time, in clock ticks, since an arbitrary point in the past (for example, system start-up time). This point does not change from one invocation of *times*( ) within the process to another. The return value may overflow the possible range of type **clock_t**. If *times*( ) fails, (**clock_t**)–1 shall be returned and *errno* set to indicate the error.

ERRORS

The *times*( ) function shall fail if:

[EOVERFLOW] The return value would overflow the range of **clock_t**.

EXAMPLES

Timing a Database Lookup

The following example defines two functions, *start_clock*( ) and *end_clock*( ), that are used to time a lookup. It also defines variables of type **clock_t** and **tms** to measure the duration of transactions. The *start_clock*( ) function saves the beginning times given by the *times*( ) function. The *end_clock*( ) function gets the ending times and prints the difference between the two times.

```
#include <sys/times.h>
#include <stdio.h>
...
void start_clock(void);
void end_clock(char *msg);
...
static clock_t st_time;
static clock_t en_time;
```



```
74293       static struct tms st_cpu;
74294       static struct tms en_cpu;
74295       ...
74296       void
74297       start_clock()
74298       {
74299           st_time = times(&st_cpu);
74300       }

74301       /* This example assumes that the result of each subtraction
74302          is within the range of values that can be represented in
74303          an integer type. */
74304       void
74305       end_clock(char *msg)
74306       {
74307           en_time = times(&en_cpu);

74308           fputs(msg,stdout);
74309           printf("Real Time: %jd, User Time %jd, System Time %jd\n",
74310               (intmax_t)(en_time - st_time),
74311               (intmax_t)(en_cpu.tms_utime - st_cpu.tms_utime),
74312               (intmax_t)(en_cpu.tms_stime - st_cpu.tms_stime));
74313       }
```

APPLICATION USAGE

Applications should use *sysconf*(_SC_CLK_TCK) to determine the number of clock ticks per second as it may vary from system to system.

RATIONALE

The accuracy of the times reported is intentionally left unspecified to allow implementations flexibility in design, from uniprocessor to multi-processor networks.

The inclusion of times of child processes is recursive, so that a parent process may collect the total times of all of its descendants. But the times of a child are only added to those of its parent when its parent successfully waits on the child. Thus, it is not guaranteed that a parent process can always see the total times of all its descendants; see also the discussion of the term ``realtime'' in *alarm( )*.

If the type **clock_t** is defined to be a signed 32-bit integer, it overflows in somewhat more than a year if there are 60 clock ticks per second, or less than a year if there are 100. There are individual systems that run continuously for longer than that. This volume of POSIX.1-2024 permits an implementation to make the reference point for the returned value be the start-up time of the process, rather than system start-up time.

The term ``charge'' in this context has nothing to do with billing for services. The operating system accounts for time used in this way. That information must be correct, regardless of how that information is used.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *exec*, *fork( )*, *sysconf( )*, *time( )*, *wait( )*, *waitid( )*

XBD **<sys/times.h>**

74338 **CHANGE HISTORY**

74339 First released in Issue 1. Derived from Issue 1 of the SVID.

74340 **Issue 7**

74341 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0371 [644] is applied.

NAME

timespec_get — get time

SYNOPSIS

```
#include <time.h>

int timespec_get(struct timespec *ts, int base);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *timespec_get*( ) function shall set the interval pointed to by *ts* to hold the current calendar time based on the specified time base.

CX If *base* is TIME_UTC, the members of *ts* shall be set to the same values as would be set by a call to *clock_gettime*(CLOCK_REALTIME, *ts*). If the number of seconds will not fit in an object of type **time_t**, the function shall return zero.

RETURN VALUE

If the *timespec_get*( ) function is successful it shall return the non-zero value *base*; otherwise, it shall return zero.

ERRORS

See DESCRIPTION.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getres( ), *time*( )

XBD **<time.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

74374 **NAME**

74375 timezone — difference from UTC and local standard time

74376 **SYNOPSIS**

74377 XSI
```
#include <time.h>

extern long timezone;
```

74379 **DESCRIPTION**

74380 Refer to *tzset( )*.

NAME

tmpfile — create a temporary file

SYNOPSIS

```
#include <stdio.h>

FILE *tmpfile(void);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *tmpfile*( ) function shall create a temporary file and open a corresponding stream. The file shall be automatically deleted when all references to the file are closed. The file shall be opened as in *fopen*( ) for update (*wb*+), except that implementations may restrict the permissions, either by clearing the file mode bits or setting them to the value S_IRUSR | S_IWUSR.

CX In some implementations, a permanent file may be left behind if the process calling *tmpfile*( ) is killed while it is processing a call to *tmpfile*( ).

An error message may be written to standard error if the stream cannot be opened.

RETURN VALUE

Upon successful completion, *tmpfile*( ) shall return a pointer to the stream of the file that is created. Otherwise, it shall return a null pointer CX and set *errno* to indicate the error.

ERRORS

The *tmpfile*( ) function shall fail if:

CX [EINTR] A signal was caught during *tmpfile*( ).

CX [EMFILE] All file descriptors available to the process are currently open.

CX [EMFILE] {STREAM_MAX} streams are currently open in the calling process.

CX [ENFILE] The maximum allowable number of files is currently open in the system.

CX [ENOSPC] The directory or file system which would contain the new file cannot be expanded.

The *tmpfile*( ) function may fail if:

CX [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

CX [ENOMEM] Insufficient storage space is available.

EXAMPLES

Creating a Temporary File

The following example creates a temporary file for update, and returns a pointer to a stream for the created file in the *fp* variable.

```
#include <stdio.h>
...
FILE *fp;

fp = tmpfile ();
```


APPLICATION USAGE

It should be possible to open at least {TMP_MAX} temporary files during the lifetime of the program (this limit may be shared with *tmpnam*( )) and there should be no limit on the number simultaneously open other than this limit and any limit on the number of open file descriptors or streams ({OPEN_MAX}, {FOPEN_MAX}, {STREAM_MAX}).

In multi-threaded applications, the *tmpfile*( ) function can leak file descriptors into child processes. Applications should instead use *mkostemp*( ) with the O_CLOEXEC or O_CLOFORK flag, or both, followed by *fdopen*( ), to avoid the leak.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fopen*( ), *mkdtemp*( ), *tmpnam*( ), *unlink*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Large File Summit extensions are added.

The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes in previous issues.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support large files.
- The [EMFILE] optional error condition is added.

The APPLICATION USAGE section is added for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #025 is applied, clarifying that implementations may restrict the permissions of the file created.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-149 is applied, adding the mandatory [EMFILE] error condition for {STREAM_MAX} streams open.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0668 [14] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0372 [678] is applied.

Issue 8

Austin Group Defects 411 and 1318 are applied, changing the APPLICATION USAGE section.

74459 Austin Group Defect 1296 is applied, removing [EOVERFLOW] from the ERRORS section.

NAME

tmpnam — create a name for a temporary file

SYNOPSIS

OB
```
#include <stdio.h>

char *tmpnam(char *s);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *tmpnam*( ) function shall generate a string that is a valid pathname that does not name an existing file. The function is potentially capable of generating {TMP_MAX} different strings, but any or all of them may already be in use by existing files and thus not be suitable return values.

The *tmpnam*( ) function generates a different string each time it is called from the same process, up to {TMP_MAX} times. If it is called more than {TMP_MAX} times, the behavior is implementation-defined.

The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls *tmpnam*( ).

If called with a null pointer argument, the *tmpnam*( ) function need not be thread-safe; however, such calls shall avoid data races with calls to *tmpnam*( ) with a non-null argument and with calls to all other functions.

RETURN VALUE

Upon successful completion, *tmpnam*( ) shall return a pointer to a string. If no suitable string can be generated, the *tmpnam*( ) function shall return a null pointer.

If the argument *s* is a null pointer, *tmpnam*( ) shall leave its result in an internal static object and return a pointer to that object. Subsequent calls to *tmpnam*( ) may modify the same object. If the argument *s* is not a null pointer, it is presumed to point to an array of at least L_tmpnam **char**s; *tmpnam*( ) shall write its result in that array and shall return the argument as its value.

ERRORS

No errors are defined.

EXAMPLES

Generating a Pathname

The following example generates a unique pathname and stores it in the array pointed to by *ptr*.

```
#include <stdio.h>
...
char pathname[L_tmpnam+1];
char *ptr;

ptr = tmpnam(pathname);
```

APPLICATION USAGE

This function only creates pathnames. It is the application's responsibility to create and remove the files.

Between the time a pathname is created and the file is opened, it is possible for some other process to create a file with the same name. Applications may find *tmpfile*( ) more useful.

Applications should use the *tmpfile*( ), *mkostemp*( ), *mkstemp*( ), or *mkdtemp*( ) functions instead of the obsolescent *tmpnam*( ) function.

RATIONALE

None.

FUTURE DIRECTIONS

The *tmpnam*( ) function may be removed in a future version, but not until after it has been removed from the ISO C standard.

SEE ALSO

fopen( ), *open*( ), *mkdtemp*( ), *tmpfile*( ), *unlink*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

Extensions beyond the ISO C standard are marked.

The normative text is updated to avoid use of the term ``must'' for application requirements.

The DESCRIPTION is expanded for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/142 is applied, updating the DESCRIPTION to allow implementations of the *tempnam*( ) function to call *tmpnam*( ).

Issue 7

Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *tmpnam*( ) function need not be thread-safe if called with a NULL parameter.

The *tmpnam*( ) function is marked obsolescent.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0669 [291] and XSH/TC1-2008/0670 [291,429] are applied.

Issue 8

Austin Group Defect 411 is applied, adding *mkostemp*( ).

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899: 2018 standard.

Austin Group Defect 1330 is applied, changing the FUTURE DIRECTIONS section and removing obsolescent interfaces.

NAME

tolower, tolower_l — transliterate uppercase characters to lowercase

SYNOPSIS

```
#include <ctype.h>

int tolower(int c);
CX int tolower_l(int c, locale_t locale);
```

DESCRIPTION

CX For *tolower*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *tolower*( ) and *tolower_l*( ) functions have as a domain a type **int**, the value of which is representable as an **unsigned char** or the value of EOF. If the argument has any other value, the behavior is undefined. If the argument of *tolower*( ) or *tolower_l*( ) represents an uppercase letter, and there exists a corresponding lowercase letter as defined by character type information in the current locale or in the locale represented by *locale*, respectively (category *LC_CTYPE*), the result shall be the corresponding lowercase letter. All other arguments in the domain are returned unchanged.

CX The behavior is undefined if the *locale* argument to *tolower_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX Upon successful completion, the *tolower*( ) and *tolower_l*( ) functions shall return the lowercase letter corresponding to the argument passed; otherwise, they shall return the argument unchanged.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setlocale( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

74574 **Issue 7**

74575 The *tolower_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API
74576 Set Part 4.

74577 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0671 [283] and XSH/TC1-2008/0672
74578 [283] are applied.

NAME

toupper, toupper_l — transliterate lowercase characters to uppercase

SYNOPSIS

```
#include <ctype.h>

int toupper(int c);
CX int toupper_l(int c, locale_t locale);
```

DESCRIPTION

CX For *toupper*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *toupper*( ) and *toupper_l*( ) functions have as a domain a type **int**, the value of which is representable as an **unsigned char** or the value of EOF. If the argument has any other value, the behavior is undefined.

CX If the argument of *toupper*( ) or *toupper_l*( ) represents a lowercase letter, and there exists a
CX corresponding uppercase letter as defined by character type information in the current locale or in the locale represented by *locale*, respectively (category *LC_CTYPE*), the result shall be the corresponding uppercase letter.

All other arguments in the domain are returned unchanged.

CX The behavior is undefined if the *locale* argument to *toupper_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX Upon successful completion, *toupper*( ) and *toupper_l*( ) shall return the uppercase letter corresponding to the argument passed; otherwise, they shall return the argument unchanged.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setlocale( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

Issue 7

SD5-XSH-ERN-181 is applied, clarifying the RETURN VALUE section.

The *toupper_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0673 [283] and XSH/TC1-2008/0674 [283] are applied.

NAME

towctrans, towctrans_l — wide-character transliteration

SYNOPSIS

```
#include <wctype.h>

wint_t towctrans(wint_t wc, wctrans_t desc);
wint_t towctrans_l(wint_t wc, wctrans_t desc,
    locale_t locale);
```

DESCRIPTION

For *towctrans*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *towctrans*( ) and *towctrans_l*( ) functions shall transliterate the wide-character code *wc* using the mapping described by *desc*.

The current setting of the *LC_CTYPE* category in the current locale or in the locale represented by *locale*, respectively, should be the same as during the call to *wctrans*( ) or *wctrans_l*( ) that returned the value *desc*.

If the value of *desc* is invalid (that is, not obtained by a call to *wctrans*( ) or *desc* is invalidated by a subsequent call to *setlocale*( ) that has affected category *LC_CTYPE*), the result is unspecified.

If the value of *desc* is invalid (that is, not obtained by a call to *wctrans_l*( ) with the same locale object *locale*) the result is unspecified.

An application wishing to check for error situations should set *errno* to 0 before calling *towctrans*( ) or *towctrans_l*( ).

If *errno* is non-zero on return, an error has occurred.

The behavior is undefined if the *locale* argument to *towctrans_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

If successful, the *towctrans*( ) and *towctrans_l*( ) functions shall return the mapped value of *wc* using the mapping described by *desc*, or the value of *wc* unchanged if *desc* is zero. Otherwise, they shall return *wc* unchanged.

ERRORS

These functions may fail if:

[EINVAL] *desc* contains an invalid transliteration descriptor.

EXAMPLES

None.

APPLICATION USAGE

The strings `"tolower"` and `"toupper"` are reserved for the standard mapping names. In the table below, the functions in the left column are equivalent to the functions in the right column.

```
towlower(wc)            towctrans(wc, wctrans("tolower"))
towlower_l(wc, locale)  towctrans_l(wc, wctrans("tolower"), locale)
towupper(wc)            towctrans(wc, wctrans("toupper"))
towupper_l(wc, locale)  towctrans_l(wc, wctrans("toupper"), locale)
```


74666 **RATIONALE**

74667 None.

74668 **FUTURE DIRECTIONS**

74669 None.

74670 **SEE ALSO**

74671 *towlower*( ), *towupper*( ), *wctrans*( )

74672 XBD **<wctype.h>**

74673 **CHANGE HISTORY**

74674 First released in Issue 5. Derived from ISO/IEC 9899: 1990/Amendment 1: 1995 (E).

74675 **Issue 6**

74676 Extensions beyond the ISO C standard are marked.

74677 **Issue 7**

74678 The *towctrans_l*( ) function is added from The Open Group Technical Standard, 2006, Extended
74679 API Set Part 4.

74680 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0675 [302], XSH/TC1-2008/0676 [283],
74681 and XSH/TC1-2008/0677 [283] are applied.

74682 **Issue 8**

74683 Austin Group Defect 1302 is applied, aligning the *towctrans*( ) function with the
74684 ISO/IEC 9899: 2018 standard.

NAME

towlower, towlower_l — transliterate uppercase wide-character code to lowercase

SYNOPSIS

```
#include <wctype.h>

wint_t towlower(wint_t wc);
CX wint_t towlower_l(wint_t wc, locale_t locale);
```

DESCRIPTION

CX For *towlower*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *towlower*( ) and *towlower_l*( ) functions have as a domain a type **wint_t**, the value of which the application shall ensure is a character representable as a **wchar_t**, and a wide-character code corresponding to a valid character in the locale used by the function or the value of WEOF. If the argument has any other value, the behavior is undefined. If the argument of *towlower*( ) or *towlower_l*( ) represents an uppercase wide-character code, and there exists a corresponding lowercase wide-character code as defined by character type information in the current locale or in the locale represented by *locale*, respectively (category *LC_CTYPE*), the result shall be the corresponding lowercase wide-character code. All other arguments in the domain are returned unchanged.

CX The behavior is undefined if the *locale* argument to *towlower_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX Upon successful completion, the *towlower*( ) and *towlower_l*( ) functions shall return the lowercase letter corresponding to the argument passed; otherwise, they shall return the argument unchanged.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setlocale( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<locale.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *towlower_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0678 [302], XSH/TC1-2008/0679 [283], and XSH/TC1-2008/0680 [283] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0373 [685] is applied.

NAME

towupper, towupper_l — transliterate lowercase wide-character code to uppercase

SYNOPSIS

```
#include <wctype.h>

wint_t towupper(wint_t wc);
CX wint_t towupper_l(wint_t wc, locale_t locale);
```

DESCRIPTION

CX For *towupper*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *towupper*( ) and *towupper_l*( ) functions have as a domain a type **wint_t**, the value of which the application shall ensure is a character representable as a **wchar_t**, and a wide-character code corresponding to a valid character in the locale used by the function or the value of WEOF. If CX the argument has any other value, the behavior is undefined. If the argument of *towupper*( ) or *towupper_l*( ) represents a lowercase wide-character code, and there exists a corresponding CX uppercase wide-character code as defined by character type information in the current locale or in the locale represented by *locale*, respectively (category *LC_CTYPE*), the result shall be the corresponding uppercase wide-character code. All other arguments in the domain are returned unchanged.

CX The behavior is undefined if the *locale* argument to *towupper_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX Upon successful completion, the *towupper*( ) and *towupper_l*( ) functions shall return the uppercase letter corresponding to the argument passed. Otherwise, they shall return the argument unchanged.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setlocale( ), *uselocale*( )

XBD Chapter 7 (on page 127), **<locale.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

The *towupper_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0681 [302], XSH/TC1-2008/0682 [283], and XSH/TC1-2008/0683 [283] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0374 [685] is applied.

NAME

trunc, truncf, truncl — round to truncated integer value

SYNOPSIS

```
#include <math.h>

double trunc(double x);
float truncf(float x);
long double truncl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall round their argument to the integer value, in floating format, nearest to but no larger in magnitude than the argument.

MX These functions may raise the inexact floating-point exception for finite non-integer arguments.

RETURN VALUE

Upon successful completion, these functions shall return the truncated integer value.

MX The returned value shall be exact, shall be independent of the current rounding direction mode, and shall have the same sign as *x*.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The integral value returned by these functions need not be expressible as an **intmax_t**. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0684 [346] is applied.

Issue 8

Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899: 2018 standard.

74832 **NAME**

74833 truncate — truncate a file to a specified length

74834 **SYNOPSIS**

74835 ```
#include <unistd.h>
```

74836 ```
int truncate(const char *path, off_t length);
```

74837 **DESCRIPTION**

74838 The *truncate*( ) function shall cause the regular file named by *path* to have a size which shall be
74839 equal to *length* bytes.

74840 If the file previously was larger than *length*, the extra data is discarded. If the file was previously
74841 shorter than *length*, its size is increased, and the extended area appears as if it were zero-filled.

74842 The application shall ensure that the process has write permission for the file.

74843 If the request would cause the file size to exceed the soft file size limit for the process, the
74844 XSI request shall fail and the implementation shall generate a SIGXFSZ signal for the thread.

74845 The *truncate*( ) function shall not modify the file offset for any open file descriptions associated
74846 with the file. Upon successful completion, *truncate*( ) shall mark for update the last data
74847 modification and last file status change timestamps of the file, and the S_ISUID and S_ISGID bits
74848 of the file mode may be cleared.

74849 **RETURN VALUE**

74850 Upon successful completion, *truncate*( ) shall return 0. Otherwise, –1 shall be returned, and *errno*
74851 set to indicate the error.

74852 **ERRORS**

74853 The *truncate*( ) function shall fail if:

74854 [EACCES] A component of the path prefix denies search permission, or write permission
74855 is denied on the file.

74856 [EFBIG] or [EINVAL]
74857 The *length* argument is greater than the maximum file size.

74858 XSI [EFBIG] The *length* argument exceeds the file size limit of the process. A SIGFSZ
74859 signal shall also be generated for the thread.

74860 [EINTR] A signal was caught during execution.

74861 [EINVAL] The *length* argument is less than 0 or the *path* argument refers to a file, other
74862 than a directory, on which this operation is not possible (for example, a FIFO
74863 or socket).

74864 [EIO] An I/O error occurred while reading from or writing to a file system.

74865 [EISDIR] The named file is a directory.

74866 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
74867 argument.

74868 [ENAMETOOLONG]
74869 The length of a component of a pathname is longer than {NAME_MAX}.

74870 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

74871 [ENOTDIR] A component of the path prefix names an existing file that is neither a
74872 directory nor a symbolic link to a directory, or the *path* argument contains at
74873 least one non-<slash> character and ends with one or more trailing <slash>

74874 characters and the last pathname component names an existing file that is
74875 neither a directory nor a symbolic link to a directory.

74876 [EROFS] The named file resides on a read-only file system.

74877 The *truncate*( ) function may fail if:

74878 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
74879 resolution of the *path* argument.

74880 [ENAMETOOLONG]
74881 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
74882 symbolic link produced an intermediate result with a length that exceeds
74883 {PATH_MAX}.

74884 **EXAMPLES**

74885 None.

74886 **APPLICATION USAGE**

74887 None.

74888 **RATIONALE**

74889 None.

74890 **FUTURE DIRECTIONS**

74891 None.

74892 **SEE ALSO**

74893 *open*( )

74894 XBD **<unistd.h>**

74895 **CHANGE HISTORY**

74896 First released in Issue 4, Version 2.

74897 **Issue 5**

74898 Moved from X/OPEN UNIX extension to BASE.

74899 Large File Summit extensions are added.

74900 **Issue 6**

74901 This reference page is split out from the *ftruncate*( ) reference page.

74902 The normative text is updated to avoid use of the term ``must'' for application requirements.

74903 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
74904 [ELOOP] error condition is added.

74905 **Issue 7**

74906 Austin Group Interpretation 1003.1-2001 #143 is applied.

74907 The *truncate*( ) function is moved from the XSI option to the Base.

74908 Changes are made related to support for finegrained timestamps.

74909 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
74910 pathname exists but is not a directory or a symbolic link to a directory.

74911 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0685 [324] is applied.

74912 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0375 [489] is applied.

Issue 8

Austin Group Defects 308 and 1087 are applied, clarifying the handling of [EFBIG] errors.

Austin Group Defect 1381 is applied, adding a second condition to the [EINVAL] error and rearranging the ERRORS section into alphabetical order.

Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error relating to the file size limit for the process.

74919 **NAME**

74920 truncf, truncl — round to truncated integer value

74921 **SYNOPSIS**

74922 ```
#include <math.h>
```

74923 ```
float truncf(float x);
```

74924 ```
long double truncl(long double x);
```

74925 **DESCRIPTION**

74926 Refer to *trunc( )*.

  
```

NAME

tsearch — search a binary search tree

SYNOPSIS

XSI

```
#include <search.h>

void *tsearch(const void *key, void **rootp,
    int (*compar)(const void *, const void *));
```

DESCRIPTION

Refer to *tdelete( )*.

NAME

tss_create — thread-specific data key creation

SYNOPSIS

```
#include <threads.h>

int tss_create(tss_t *key, tss_dtor_t dtor);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *tss_create*( ) function shall create a thread-specific storage pointer with destructor *dtor*, which can be null.

A null pointer value shall be associated with the newly created key in all existing threads. Upon subsequent thread creation, the value associated with all keys shall be initialized to a null pointer value in the new thread.

Destructors associated with thread-specific storage shall not be invoked at process termination.

The behavior is undefined if the *tss_create*( ) function is called from within a destructor.

CX The *tss_create*( ) function shall not be affected if the calling thread executes a signal handler during the call.

RETURN VALUE

If the *tss_create*( ) function is successful, it shall set the thread-specific storage pointed to by *key* to a value that uniquely identifies the newly created pointer and shall return `thrd_success`; otherwise, `thrd_error` shall be returned and the thread-specific storage pointed to by *key* has an indeterminate value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *tss_create*( ) function performs no implicit synchronization. It is the responsibility of the application writer to ensure that it is called exactly once per key before use of the key.

RATIONALE

If the value associated with a key needs to be updated during the lifetime of the thread, it may be necessary to release the storage associated with the old value before the new value is bound. Although the *tss_set*( ) function could do this automatically, this feature is not needed often enough to justify the added complexity. Instead, the application is responsible for freeing the stale storage:

```
old = tss_get(key);
new = allocate();
destructor(old);
tss_set(key, new);
```

There is no notion of a destructor-safe function. If an application does not call *thrd_exit*( ) or *pthread_exit*( ) from a signal handler, or if, while calling async-unsafe functions, it blocks any signal whose handler might call *thrd_exit*( ) or *pthread_exit*( ), all functions can be safely called from destructors.

The *tss_create*( ) function is not affected by signal handlers for the reasons stated in XRAT Section B.2.3 (on page 3742).

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_exit( ), *pthread_key_create*( ), *thrd_exit*( ), *tss_delete*( ), *tss_get*( )

XBD **<threads.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

NAME

tss_delete — thread-specific data key deletion

SYNOPSIS

```
#include <threads.h>

void tss_delete(tss_t key);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *tss_delete*( ) function shall release any resources used by the thread-specific storage identified by *key*. The thread-specific data values associated with *key* need not be null at the time *tss_delete*( ) is called. It is the responsibility of the application to free any application storage or perform any cleanup actions for data structures related to the deleted key or associated thread-specific data in any threads; this cleanup can be done either before or after *tss_delete*( ) is called.

The application shall ensure that the *tss_delete*( ) function is only called with a value for *key* that was returned by a call to *tss_create*( ) before the thread commenced executing destructors.

If *tss_delete*( ) is called while another thread is executing destructors, whether this will affect the number of invocations of the destructor associated with *key* on that thread is unspecified.

The *tss_delete*( ) function shall be callable from within destructor functions. Calling *tss_delete*( ) shall not result in the invocation of any destructors. Any destructor function that was associated with *key* shall no longer be called upon thread exit.

Any attempt to use *key* following the call to *tss_delete*( ) results in undefined behavior.

CX The *tss_delete*( ) function shall not be affected if the calling thread executes a signal handler during the call.

RETURN VALUE

This function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

A thread-specific data key deletion function has been included in order to allow the resources associated with an unused thread-specific data key to be freed. Unused thread-specific data keys can arise, among other scenarios, when a dynamically loaded module that allocated a key is unloaded.

Conforming applications are responsible for performing any cleanup actions needed for data structures associated with the key to be deleted, including data referenced by thread-specific data values. No such cleanup is done by *tss_delete*( ). In particular, destructor functions are not called. See the RATIONALE for *pthread_key_delete*( ) for the reasons for this division of responsibility.

The *tss_delete*( ) function is not affected by signal handlers for the reasons stated in XRAT Section B.2.3 (on page 3742).

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_key_delete( ), *tss_create*( )

XBD **<threads.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

tss_get, tss_set — thread-specific data management

SYNOPSIS

```
#include <threads.h>

void *tss_get(tss_t key);
int   tss_set(tss_t key, void *val);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *tss_get*( ) function shall return the value for the current thread held in the thread-specific storage identified by *key*.

The *tss_set*( ) function shall set the value for the current thread held in the thread-specific storage identified by *key* to *val*. This action shall not invoke the destructor associated with the *key* on the value being replaced.

The application shall ensure that the *tss_get*( ) and *tss_set*( ) functions are only called with a value for *key* that was returned by a call to *tss_create*( ) before the thread commenced executing destructors.

The effect of calling *tss_get*( ) or *tss_set*( ) after *key* has been deleted with *tss_delete*( ) is undefined.

CX Both *tss_get*( ) and *tss_set*( ) can be called from a thread-specific data destructor function. A call to *tss_get*( ) for the thread-specific data key being destroyed shall return a null pointer, unless the value is changed (after the destructor starts) by a call to *tss_set*( ). Calling *tss_set*( ) from a thread-specific data destructor function may result either in lost storage (after at least {PTHREAD_DESTRUCTOR_ITERATIONS} attempts at destruction) or in an infinite loop.

These functions shall not be affected if the calling thread executes a signal handler during the call.

RETURN VALUE

The *tss_get*( ) function shall return the value for the current thread. If no thread-specific data value is associated with *key*, then a null pointer shall be returned.

The *tss_set*( ) function shall return `thrd_success` on success or `thrd_error` if the request could not be honored.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

These functions are not affected by signal handlers for the reasons stated in XRAT Section B.2.3 (on page 3742).

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_getspecific( ), *tss_create*( )

XBD **<threads.h>**

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

ttyname, ttyname_r — find the pathname of a terminal

SYNOPSIS

```
#include <unistd.h>

char *ttyname(int fildes);
int ttyname_r(int fildes, char *name, size_t namesize);
```

DESCRIPTION

The *ttyname*( ) function shall return a pointer to a string containing a null-terminated pathname of the terminal associated with file descriptor *fildes*. The application shall not modify the string returned. The returned pointer might be invalidated or the string content might be overwritten by a subsequent call to *ttyname*( ). The returned pointer and the string content might also be invalidated if the calling thread is terminated.

The *ttyname*( ) function need not be thread-safe.

The *ttyname_r*( ) function shall store the null-terminated pathname of the terminal associated with the file descriptor *fildes* in the character array referenced by *name*. The array is *namesize* characters long and should have space for the name and the terminating null character. The maximum length of the terminal name shall be {TTY_NAME_MAX}.

RETURN VALUE

Upon successful completion, *ttyname*( ) shall return a pointer to a string. Otherwise, a null pointer shall be returned and *errno* set to indicate the error.

If successful, the *ttyname_r*( ) function shall return zero. Otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

[ENOTTY] The file associated with the *fildes* argument is not a terminal.

The *ttyname_r*( ) function shall fail if:

[ERANGE] The value of *namesize* is smaller than the length of the string to be returned including the terminating null character.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The term ``terminal'' is used instead of the historical term ``terminal device'' in order to avoid a reference to an undefined term.

The thread-safe version places the terminal name in a user-supplied buffer and returns a non-zero value if it fails. The non-thread-safe version may return the name in a static data area that may be overwritten by each call.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The *ttyname_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *ttyname*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *ttyname_r*( ) function is marked as part of the Thread-Safe Functions option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The statement that *errno* is set on error is added.
- The [EBADF] and [ENOTTY] optional error conditions are added.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

The *ttyname_r*( ) function is moved from the Thread-Safe Functions option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0686 [75] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0376 [656] is applied.

Issue 8

Austin Group Defect 398 is applied, combining the duplicated [EBADF] and [ENOTTY] errors and changing the [ERANGE] error from ``may fail'' to ``shall fail''.

NAME

twalk — traverse a binary search tree

SYNOPSIS

XSI

```
#include <search.h>

void twalk(const void *root,
    void (*action)(const void *, VISIT, int ));
```

DESCRIPTION

Refer to *tdelete( )*.

NAME

daylight, timezone, tzname, tzset — set timezone conversion information

SYNOPSIS

```
#include <time.h>

extern int daylight;
extern long timezone;
extern char *tzname[2];
void tzset(void);
```

DESCRIPTION

The *tzset*( ) function shall use the value of the environment variable *TZ* to set time conversion information used by *ctime*( ), *localtime*( ), *mktime*( ), and *strftime*( ). If *TZ* is absent from the environment, implementation-defined default timezone information shall be used.

The *tzset*( ) function shall set the external variable *tzname* as follows:

```
tzname[0] = "std";
tzname[1] = "dst";
```

where *std* and *dst* are as described in XBD Chapter 8 (on page 167).

The *tzset*( ) function also shall set the external variable *daylight* to 0 if Daylight Saving Time conversions should never be applied for the timezone in use; otherwise, non-zero. The external variable *timezone* shall be set to the difference, in seconds, between Coordinated Universal Time (UTC) and local standard time.

If a thread accesses *tzname*, *daylight*, or *timezone* directly while another thread is in a call to *tzset*( ), or to any function that is required or allowed to set timezone information as if by calling *tzset*( ), the behavior is undefined.

RETURN VALUE

The *tzset*( ) function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

Example *TZ* variables and their timezone differences are given in the table below:

TZ	*timezone*
EST5EDT	5*60*60
GMT0	0*60*60
JST-9	–9*60*60
MET-1MEST	–1*60*60
MST7MDT	7*60*60
PST8PDT	8*60*60

APPLICATION USAGE

Since the *ctime*( ), *localtime*( ), *mktime*( ), *strftime*( ), and *strftime_l*( ) functions are required to set timezone information as if by calling *tzset*( ), there is no need for an explicit *tzset*( ) call before using these functions. However, portable applications should call *tzset*( ) explicitly before using *localtime_r*( ) because setting timezone information is optional for that function.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ctime( ), *localtime*( ), *mktime*( ), *strftime*( )

XBD Chapter 8 (on page 167), **<time.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The example is corrected.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0377 [880] is applied.

Issue 8

Austin Group Defect 1253 is applied, changing ``Daylight Savings'' to ``Daylight Saving''.

Austin Group Defect 1410 is applied, removing the *ctime_r*( ) function.

NAME

umask — set and get the file mode creation mask

SYNOPSIS

```
#include <sys/stat.h>

mode_t umask(mode_t cmask);
```

DESCRIPTION

The *umask*( ) function shall set the file mode creation mask of the process to *cmask* and return the previous value of the mask. Only the file permission bits of *cmask* (see **<sys/stat.h>**) shall be used; XSI the S_ISVTX bit shall be ignored, and the meaning of the other bits is implementation-defined.

The file mode creation mask of the process is used to turn off permission bits in the *mode* argument supplied during calls to the following functions:

- *open*( ), *openat*( ), *creat*( ), *mkdir*( ), *mkdirat*( ), *mkfifo*( ), and *mkfifoat*( )
- XSI *mknod*( ), *mknodat*( )
- MSG *mq_open*( )
- *sem_open*( )

Permission bit positions that are set in *cmask* are cleared in the mode of the created file.

RETURN VALUE

The file permission bits in the value returned by *umask*( ) shall be the previous value of the file mode creation mask. XSI The S_ISVTX bit in the returned value shall be clear. The state of any other bits in the returned value is unspecified, except that a subsequent call to *umask*( ) with the returned value as *cmask* shall leave the state of the mask the same as its state before the first call, including any unspecified use of those bits.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Unsigned argument and return types for *umask*( ) were proposed. The return type and the argument were both changed to **mode_t**.

Historical implementations have made use of additional bits in *cmask* for their implementation-defined purposes. The addition of the text that the meaning of other bits of the field is implementation-defined permits these implementations to conform to this volume of POSIX.1-2024.

FUTURE DIRECTIONS

None.

SEE ALSO

creat( ), *exec*, *mkdir*( ), *mkfifo*( ), *mknod*( ), *mq_open*( ), *open*( ), *sem_open*( )

XBD **<sys/stat.h>**, **<sys/types.h>**

75256 **CHANGE HISTORY**

75257 First released in Issue 1. Derived from Issue 1 of the SVID.

75258 **Issue 6**

75259 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

75260 The following new requirements on POSIX implementations derive from alignment with the
75261 Single UNIX Specification:

75262 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
75263 required for conforming implementations of previous POSIX specifications, it was not
75264 required for UNIX applications.

75265 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/143 is applied, adding the *mknod*( ),
75266 *mq_open*( ), and *sem_open*( ) functions to the DESCRIPTION and SEE ALSO sections.

75267 **Issue 8**

75268 Austin Group Defect 1522 is applied, adding requirements relating to the S_ISVTX bit.

NAME

uname — get the name of the current system

SYNOPSIS

```
#include <sys/utsname.h>

int uname(struct utsname *name);
```

DESCRIPTION

The *uname*( ) function shall store information identifying the current system in the structure pointed to by *name*.

The *uname*( ) function uses the **utsname** structure defined in **<sys/utsname.h>**.

The *uname*( ) function shall return a string naming the current system in the character array *sysname*. Similarly, *nodename* shall contain the name of this node within an implementation-defined communications network. The arrays *release* and *version* shall further identify the operating system. The array *machine* shall contain a name that identifies the hardware that the system is running on.

The format of each member is implementation-defined.

RETURN VALUE

Upon successful completion, a non-negative value shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The inclusion of the *nodename* member in this structure does not imply that it is sufficient information for interfacing to communications networks.

RATIONALE

The values of the structure members are not constrained to have any relation to the version of this volume of POSIX.1-2024 implemented in the operating system. An application should instead depend on _POSIX_VERSION and related constants defined in **<unistd.h>**.

This volume of POSIX.1-2024 does not define the sizes of the members of the structure and permits them to be of different sizes, although most implementations define them all to be the same size: eight bytes plus one byte for the string terminator. That size for *nodename* is not enough for use with many networks.

The *uname*( ) function originated in System III, System V, and related implementations, and it does not exist in Version 7 or 4.3 BSD. The values it returns are set at system compile time in those historical implementations.

4.3 BSD has *gethostname*( ) and *gethostid*( ), which return a symbolic name and a numeric value, respectively. There are related *sethostname*( ) and *sethostid*( ) functions that are used to set the values the other two functions return. The former functions are included in this specification, the latter are not.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD **<sys/utsname.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

NAME

ungetc — push byte back into input stream

SYNOPSIS

```
#include <stdio.h>

int ungetc(int c, FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *ungetc*( ) function shall push the byte specified by *c* (converted to an **unsigned char**) back onto the input stream pointed to by *stream*. The pushed-back bytes shall be returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening CX call (with the stream pointed to by *stream*) to a file-positioning function (*fseek*( ), *fseeko*( ), CX *fsetpos*( ), or *rewind*( )) or *fflush*( ) shall discard any pushed-back bytes for the stream. The external storage corresponding to the stream shall be unchanged.

One byte of push-back shall be provided. If *ungetc*( ) is called too many times on the same stream without an intervening read or file-positioning operation on that stream, the operation may fail.

If the value of *c* equals that of the macro EOF, the operation shall fail and the input stream shall be left unchanged.

A successful call to *ungetc*( ) shall clear the end-of-file indicator for the stream. The file-position indicator for the stream shall be decremented by each successful call to *ungetc*( ); if its value was 0 before a call, its value is unspecified after the call. The value of the file-position indicator after all pushed-back bytes have been read shall be the same as it was before the bytes were pushed back.

RETURN VALUE

Upon successful completion, *ungetc*( ) shall return the byte pushed back after conversion. Otherwise, it shall return EOF.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The ISO C standard includes the text ``The value of the file position indicator for the stream after reading or discarding all pushed-back characters shall be the same as it was before the characters were pushed back.'' POSIX.1 omits ``or discarding'' from this because it is redundant—in the ISO C standard the discarding is done by file positioning functions and does not affect the position set by those functions. In particular, a relative seek using *fseek*( ) or *fseeko*( ) with SEEK_CUR adjusts the position relative to the position on entry to the function, not the position after the pushed-back bytes have been discarded. POSIX.1 also requires *fflush*( ) to discard pushed back bytes in situations where the ISO C standard says the behavior of *fflush*( ) is undefined.

FUTURE DIRECTIONS

The ISO C standard states that the use of *ungetc*( ) on a binary stream where the file position indicator is zero prior to the call is an obsolescent feature. In POSIX.1 there is no distinction between binary and text streams, so this applies to all streams. This feature may be removed in a future version of this standard.

SEE ALSO

Section 2.5 (on page 521), *fseek*( ), *getc*( ), *fsetpos*( ), *read*( ), *rewind*( ), *setbuf*( )

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0687 [87,93], XSH/TC1-2008/0688 [87], and XSH/TC1-2008/0689 [14] are applied.

Issue 8

Austin Group Defect 701 is applied, clarifying how the file-position indicator for the stream is updated.

Austin Group Defect 1302 is applied, changing the FUTURE DIRECTIONS section.

NAME

ungetwc — push wide-character code back into the input stream

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

wint_t ungetwc(wint_t wc, FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *ungetwc*( ) function shall push the character corresponding to the wide-character code specified by *wc* back onto the input stream pointed to by *stream*. The pushed-back characters shall be returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening call (with the stream pointed to by *stream*) to a file-positioning function (*fseek*( ), *fseeko*( ), *fsetpos*( ), or *rewind*( )) CX or *fflush*( ) shall discard any pushed-back characters for the stream. The external storage corresponding to the stream is unchanged.

At least one character of push-back shall be provided. If *ungetwc*( ) is called too many times on the same stream without an intervening read or file-positioning operation on that stream, the operation may fail.

If the value of *wc* equals that of the macro WEOF, the operation shall fail and the input stream shall be left unchanged.

A successful call to *ungetwc*( ) shall clear the end-of-file indicator for the stream. The value of the file-position indicator for the stream after a successful call to *ungetwc*( ) is unspecified until all pushed-back wide characters are read or discarded; its value after all pushed-back wide characters have been read shall be the same as it was before the wide characters were pushed back.

RETURN VALUE

Upon successful completion, *ungetwc*( ) shall return the wide-character code corresponding to the pushed-back character. Otherwise, it shall return WEOF.

ERRORS

The *ungetwc*( ) function may fail if:

CX [EILSEQ] An invalid character sequence is detected, or a wide-character code does not correspond to a valid character.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The ISO C standard includes the text ``The value of the file position indicator for the stream after reading or discarding all pushed-back wide characters shall be the same as it was before the wide characters were pushed back.'' POSIX.1 omits ``or discarding'' from this because it is redundant—in the ISO C standard the discarding is done by file positioning functions and does not affect the position set by those functions. In particular, a relative seek using *fseek*( ) or *fseeko*( ) with SEEK_CUR adjusts the position relative to the position on entry to the function, not the position after the pushed-back wide characters have been discarded. POSIX.1 also requires *fflush*( ) to discard pushed back wide characters in situations where the ISO C standard says the

behavior of *fflush*( ) is undefined.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fseek*( ), *fsetpos*( ), *read*( ), *rewind*( ), *setbuf*( )

XBD **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

The Optional Header (OH) marking is removed from **<stdio.h>**.

Issue 6

The [EILSEQ] optional error condition is marked CX.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0690 [87,93], XSH/TC1-2008/0691 [87], and XSH/TC1-2008/0692 [14] are applied.

Issue 8

Austin Group Defect 701 is applied, clarifying how the file-position indicator for the stream is updated.

Austin Group Defect 1374 is applied, correcting a conflict with the ISO C standard regarding the value of the file-position indicator for the stream after a successful call to *ungetwc*( ).

NAME

unlink, unlinkat — remove a directory entry

SYNOPSIS

```
#include <unistd.h>

int unlink(const char *path);
```

OH

```
#include <fcntl.h>

int unlinkat(int fd, const char *path, int flag);
```

DESCRIPTION

The *unlink*( ) function shall remove the directory entry named by *path* and shall decrement the link count of the file referenced by the directory entry. If *path* names a symbolic link, *unlink*( ) shall remove the symbolic link and shall not affect any file named by the contents of the symbolic link.

When the file's link count becomes 0 and no process has a reference to the file via an open file descriptor or a memory mapping (see *mmap*( )), the space occupied by the file shall be freed and the file shall no longer be accessible. If one or more processes have such a reference to the file when the last link is removed, the link shall be removed before *unlink*( ) returns, but the removal of the file contents shall be postponed until there are no such references to the file. When the space occupied by the file has been freed, the file's serial number (*st_ino*), and therefore the file identity (see XBD **<sys/stat.h>**), shall become available for reuse.

The *path* argument shall not name a directory unless the process has appropriate privileges and the implementation supports using *unlink*( ) on directories.

Upon successful completion, *unlink*( ) shall mark for update the last data modification and last file status change timestamps of the parent directory. Also, if the file's link count is not 0, the last file status change timestamp of the file shall be marked for update.

The *unlinkat*( ) function shall be equivalent to the *unlink*( ) or *rmdir*( ) function except in the case where *path* specifies a relative path. In this case the directory entry to be removed is determined relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in **<fcntl.h>**:

AT_REMOVEDIR
: Remove the directory entry specified by *fd* and *path* as a directory, not a normal file.

If *unlinkat*( ) is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used and the behavior shall be identical to a call to *unlink*( ) or *rmdir*( ) respectively, depending on whether or not the AT_REMOVEDIR bit is set in *flag*.

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return –1 and set *errno* to indicate the error. If –1 is returned, the named file shall not be changed.

ERRORS

These functions shall fail and shall not unlink the file if:

[EACCES] Search permission is denied for a component of the path prefix, or write permission is denied on the directory containing the directory entry to be removed.

[EBUSY] The file named by the *path* argument cannot be unlinked because it is being used by the system or another process and the implementation considers this an error.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG] The length of a component of a pathname is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

XSI [EPERM] or [EACCES] The S_ISVTX flag is set on the directory containing the file referred to by the *path* argument and the process does not satisfy the criteria specified in XBD Section 4.5 (on page 96).

[EROFS] The directory entry to be unlinked is part of a read-only file system.

The *unlink*( ) function shall fail and shall not unlink the file if:

[EPERM] The file named by *path* is a directory, and either the calling process does not have appropriate privileges or the implementation prohibits using *unlink*( ) on directories.

The *unlinkat*( ) function shall fail and shall not unlink the file if:

[EACCES] The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

[EBADF] The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

[ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

[EEXIST] or [ENOTEMPTY] The *flag* parameter has the AT_REMOVEDIR bit set and the *path* argument names a directory that is not an empty directory, or there are hard links to the directory other than dot or a single entry in dot-dot.

[ENOTDIR] The *flag* parameter has the AT_REMOVEDIR bit set and *path* does not name a directory.

[EPERM] The file named by *path* is a directory, the *flag* parameter does not have the AT_REMOVEDIR bit set, and either the calling process does not have appropriate privileges or the implementation prohibits using *unlink*( ) on directories.

These functions may fail and not unlink the file if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

[ETXTBSY] The entry to be unlinked is the last directory entry to a pure procedure (shared text) file that is being executed.

The *unlinkat*( ) function may fail and not unlink the file if:

[EINVAL] The value of the *flag* argument is not valid.

EXAMPLES

Removing a Link to a File

The following example shows how to remove a link to a file named **/home/cnd/mod1** by removing the entry named **/modules/pass1**.

```
#include <unistd.h>

char *path = "/modules/pass1";
int   status;
...
status = unlink(path);
```

Checking for an Error

The following example fragment creates a temporary password lock file named **LOCKFILE**, which is defined as **/etc/ptmp**, and gets a file descriptor for it. If the file cannot be opened for writing, *unlink*( ) is used to remove the link between the file descriptor and **LOCKFILE**.

```
#include <sys/types.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <sys/stat.h>

#define LOCKFILE "/etc/ptmp"

int pfd;  /* Integer for file descriptor returned by open call. */
FILE *fpfd;  /* File pointer for use in putpwent(). */
...
/* Open password Lock file. If it exists, this is an error. */
if ((pfd = open(LOCKFILE, O_WRONLY| O_CREAT | O_EXCL, S_IRUSR
    | S_IWUSR | S_IRGRP | S_IROTH)) == -1)  {
    fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
    exit(1);
}

/* Lock file created; proceed with fdopen of lock file so that
   putpwent() can be used.
```

```
 */
if ((fpfd = fdopen(pfd, "w")) == NULL) {
    close(pfd);
    unlink(LOCKFILE);
    exit(1);
}
```

Replacing Files

The following example fragment uses *unlink*( ) to discard links to files, so that they can be replaced with new versions of the files. The first call removes the link to **LOCKFILE** if an error occurs. Successive calls remove the links to **SAVEFILE** and **PASSWDFILE** so that new links can be created, then removes the link to **LOCKFILE** when it is no longer needed.

```
#include <sys/types.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <sys/stat.h>

#define LOCKFILE "/etc/ptmp"
#define PASSWDFILE "/etc/passwd"
#define SAVEFILE "/etc/opasswd"
...
/* If no change was made, assume error and leave passwd unchanged. */
if (!valid_change) {
    fprintf(stderr, "Could not change password for user %s\n", user);
    unlink(LOCKFILE);
    exit(1);
}

/* Change permissions on new password file. */
chmod(LOCKFILE, S_IRUSR | S_IRGRP | S_IROTH);

/* Remove saved password file. */
unlink(SAVEFILE);

/* Save current password file. */
link(PASSWDFILE, SAVEFILE);

/* Remove current password file. */
unlink(PASSWDFILE);

/* Save new password file as current password file. */
link(LOCKFILE,PASSWDFILE);

/* Remove lock file. */
unlink(LOCKFILE);

exit(0);
```

APPLICATION USAGE

Applications should use *rmdir*( ) to remove a directory.

RATIONALE

Unlinking a directory is restricted to the superuser in many historical implementations for reasons given in *link*( ) (see also *rename*( )).

The meaning of [EBUSY] in historical implementations is ``mount point busy''. Since this volume of POSIX.1-2024 does not cover the system administration concepts of mounting and unmounting, the description of the error was changed to ``resource busy''. (This meaning is used by some device drivers when a second process tries to open an exclusive use device.) The wording is also intended to allow implementations to refuse to remove a directory if it is the root or current working directory of any process.

The standard developers reviewed TR 24715-2006 and noted that LSB-conforming implementations may return [EISDIR] instead of [EPERM] when unlinking a directory. A change to permit this behavior by changing the requirement for [EPERM] to [EPERM] or [EISDIR] was considered, but decided against since it would break existing strictly conforming and conforming applications. Applications written for portability to both POSIX.1-2024 and the LSB should be prepared to handle either error code.

The purpose of the *unlinkat*( ) function is to remove directory entries in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *unlink*( ), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *unlinkat*( ) function it can be guaranteed that the removed directory entry is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

close( ), *link*( ), *remove*( ), *rename*( ), *rmdir*( ), *symlink*( )

XBD Section 4.5 (on page 96), **<fcntl.h>**, **<sys/stat.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The [EBUSY] error is added to the optional part of the ERRORS section.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the effect is specified if *path* specifies a symbolic link.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [ETXTBSY] optional error condition is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

Austin Group Interpretation 1003.1-2001 #181 is applied, updating the requirements for operations when the S_ISVTX bit is set.

Text arising from the LSB Conflicts TR is added to the RATIONALE about the use of [EPERM] and [EISDIR].

The *unlinkat*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

Changes are made to allow a directory to be opened for searching.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0693 [461], XSH/TC1-2008/0694 [324], XSH/TC1-2008/0695 [278], and XSH/TC1-2008/0696 [278] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0378 [873], XSH/TC2-2008/0379 [591], XSH/TC2-2008/0380 [817], and XSH/TC2-2008/0381 [817] are applied.

Issue 8

Austin Group Defect 1314 is applied, clarifying that file identities become available for reuse after the space occupied by the file has been freed.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1380 is applied, changing text using the term ``link'' in line with its updated definition.

Austin Group Defect 1385 is applied, clarifying that the file contents are not removed until there are no references to the file via open file descriptors or memory mappings.

Austin Group Defect 1574 is applied, splitting the [EPERM] error into separate entries for *unlink*( ) and *unlinkat*( ).

NAME

unlockpt — unlock a pseudo-terminal manager/subsidiary pair

SYNOPSIS

XSI
```
#include <stdlib.h>

int unlockpt(int fildes);
```

DESCRIPTION

The *unlockpt*( ) function shall unlock the subsidiary pseudo-terminal device associated with the manager device to which *fildes* refers.

Conforming applications shall ensure that they call *unlockpt*( ) before opening the subsidiary side of a pseudo-terminal device.

RETURN VALUE

Upon successful completion, *unlockpt*( ) shall return 0. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *unlockpt*( ) function may fail if:

[EBADF] The *fildes* argument is not a file descriptor open for writing.

[EINVAL] The *fildes* argument is not associated with a manager pseudo-terminal device.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

See the RATIONALE section for *posix_openpt*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

grantpt( ), *open*( ), *posix_openpt*( ), *ptsname*( )

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0697 [96] is applied.

Issue 8

Austin Group Defect 1466 is applied, changing the terminology used for pseudo-terminal devices.

NAME

unsetenv — remove an environment variable

SYNOPSIS

CX
```
#include <stdlib.h>

int unsetenv(const char *name);
```

DESCRIPTION

The *unsetenv*( ) function shall remove an environment variable from the environment of the calling process. The *name* argument points to a string, which is the name of the variable to be removed. The named argument shall not contain an '=' character. If the named variable does not exist in the current environment, the environment shall be unchanged and the function is considered to have completed successfully.

The *unsetenv*( ) function shall update the list of pointers to which *environ* points.

The *unsetenv*( ) function need not be thread-safe.

RETURN VALUE

Upon successful completion, zero shall be returned. Otherwise, –1 shall be returned, *errno* set to indicate the error, and the environment shall be unchanged.

ERRORS

The *unsetenv*( ) function shall fail if:

[EINVAL] The *name* argument points to an empty string, or points to a string containing an '=' character.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to the RATIONALE section in *setenv*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

getenv( ), *setenv*( )

XBD **<stdlib.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0698 [167] and XSH/TC1-2008/0699 [185] are applied.

NAME

uselocale — use locale in current thread

SYNOPSIS

CX
```
#include <locale.h>

locale_t uselocale(locale_t newloc);
```

DESCRIPTION

The *uselocale*( ) function shall set or query the current locale for the calling thread.

The value for the *newloc* argument shall be one of the following:

1. A value returned by the *newlocale*( ) or *duplocale*( ) functions
2. The special locale object descriptor LC_GLOBAL_LOCALE
3. (**locale_t**)0

If the *newloc* argument is (**locale_t**)0, the current locale shall not be changed; this value can be used to query the current locale setting. If the *newloc* argument is LC_GLOBAL_LOCALE, any thread-local locale for the calling thread shall be uninstalled; the thread shall again use the global locale as the current locale, and changes to the global locale shall affect the thread. Otherwise, the locale represented by *newloc* shall be installed as a thread-local locale to be used as the current locale for the calling thread.

Once the *uselocale*( ) function has been called to install a thread-local locale, the behavior of every interface using data from the current locale shall be affected for the calling thread. The current locale for other threads shall remain unchanged.

RETURN VALUE

Upon successful completion, the *uselocale*( ) function shall return a handle for the thread-local locale that was in use as the current locale for the calling thread on entry to the function, or LC_GLOBAL_LOCALE if no thread-local locale was in use. Otherwise, *uselocale*( ) shall return (**locale_t**)0 and set *errno* to indicate the error.

ERRORS

The *uselocale*( ) function may fail if:

[EINVAL] *newloc* is not a valid locale object and is not (**locale_t**)0.

EXAMPLES

None.

APPLICATION USAGE

Unlike the *setlocale*( ) function, the *uselocale*( ) function does not allow replacing some locale categories only. Applications that need to install a locale which differs only in a few categories must use *newlocale*( ) to change a locale object equivalent to the currently used locale and install it.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

duplocale( ), *freelocale*( ), *getlocalename_l*( ), *newlocale*( ), *setlocale*( )

XBD **<locale.h>**

CHANGE HISTORY

First released in Issue 7.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0700 [290] and XSH/TC1-2008/0701 [334] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0382 [582] is applied.

Issue 8

Austin Group Defect 1220 is applied, adding *getlocalename_l*( ) to the SEE ALSO section.

75801 **NAME**

75802 utimensat, utimes — set file access and modification times

75803 **SYNOPSIS**

75804 ```
#include <sys/stat.h>
```

75805 ```
int utimensat(int fd, const char *path, const struct timespec times[2],
```

75806 ```
    int flag);
```

75807 XSI ```
#include <sys/time.h>
```

75808 ```
int utimes(const char *path, const struct timeval times[2]);
```

75809 **DESCRIPTION**

75810 Refer to *futimens( )*.

  
```

75811 **NAME**

75812 va_arg, va_copy, va_end, va_start — handle variable argument list

75813 **SYNOPSIS**

75814 `#include <stdarg.h>`

75815 `type va_arg(va_list ap, type);`
75816 `void va_copy(va_list dest, va_list src);`
75817 `void va_end(va_list ap);`
75818 `void va_start(va_list ap, argN);`

75819 **DESCRIPTION**

75820 Refer to XBD **<stdarg.h>**

NAME

vasprintf — format output of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>

int vasprintf(char **restrict ptr, const char *restrict format,
    va_list ap);
```

DESCRIPTION

Refer to *vfprintf( )*.

NAME

vasprintf, vdprintf, vfprintf, vprintf, vsnprintf, vsprintf — format output of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>

CX  int vasprintf(char **restrict ptr, const char *restrict format,
        va_list ap);
    int vdprintf(int fildes, const char *restrict format, va_list ap);
    int vfprintf(FILE *restrict stream, const char *restrict format,
        va_list ap);
    int vprintf(const char *restrict format, va_list ap);
    int vsnprintf(char *restrict s, size_t n, const char *restrict format,
        va_list ap);
    int vsprintf(char *restrict s, const char *restrict format, va_list ap);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *vasprintf*( ), *vdprintf*( ), *vfprintf*( ), *vprintf*( ), *vsnprintf*( ), and *vsprintf*( ) functions shall be CX equivalent to the *asprintf*( ), *dprintf*( ), *fprintf*( ), *printf*( ), *snprintf*( ), and *sprintf*( ) functions respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by **<stdarg.h>**.

These functions shall not invoke the *va_end* macro. As these functions invoke the *va_arg* macro, the value of *ap* after the return is unspecified.

RETURN VALUE

Refer to *fprintf*( ).

ERRORS

Refer to *fprintf*( ).

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions should call *va_end*(*ap*) afterwards to clean up.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fprintf*( )

XBD **<stdarg.h>**, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The *vsnprintf*( ) function is added.

Issue 6

The *vfprintf*( ), *vprintf*( ), *vsnprintf*( ), and *vsprintf*( ) functions are updated for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

The *vdprintf*( ) function is added to complement the *dprintf*( ) function from The Open Group Technical Standard, 2006, Extended API Set Part 1.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0703 [14] is applied.

Issue 8

Austin Group Defect 1496 is applied, adding the *vasprintf*( ) function.

NAME

vfscanf, vscanf, vsscanf — format input of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>

int vfscanf(FILE *restrict stream, const char *restrict format,
    va_list arg);
int vscanf(const char *restrict format, va_list arg);
int vsscanf(const char *restrict s, const char *restrict format,
    va_list arg);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *vscanf*( ), *vfscanf*( ), and *vsscanf*( ) functions shall be equivalent to the *scanf*( ), *fscanf*( ), and *sscanf*( ) functions, respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined in the **<stdarg.h>** header. These functions shall not invoke the *va_end* macro. As these functions invoke the *va_arg* macro, the value of *ap* after the return is unspecified.

RETURN VALUE

Refer to *fscanf*( ).

ERRORS

Refer to *fscanf*( ).

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions should call *va_end*(*ap*) afterwards to clean up.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fscanf*( )

XBD **<stdarg.h>**, **<stdio.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0704 [14] is applied.

NAME

vfwprintf, vswprintf, vwprintf — wide-character formatted output of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vfwprintf(FILE *restrict stream, const wchar_t *restrict format,
    va_list arg);
int vswprintf(wchar_t *restrict ws, size_t n,
    const wchar_t *restrict format, va_list arg);
int vwprintf(const wchar_t *restrict format, va_list arg);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *vfwprintf*( ), *vswprintf*( ), and *vwprintf*( ) functions shall be equivalent to *fwprintf*( ), *swprintf*( ), and *wprintf*( ) respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by **<stdarg.h>**.

These functions shall not invoke the *va_end* macro. However, as these functions do invoke the *va_arg* macro, the value of *ap* after the return is unspecified.

RETURN VALUE

Refer to *fwprintf*( ).

ERRORS

Refer to *fwprintf*( ).

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions should call *va_end*(*ap*) afterwards to clean up.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fwprintf*( )

XBD **<stdarg.h>**, **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The *vfwprintf*( ), *vswprintf*( ), and *vwprintf*( ) prototypes are updated for alignment with the ISO/IEC 9899:1999 standard.

75962 **Issue 7**
75963 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0705 [14] is applied.

NAME

vfwscanf, vswscanf, vwscanf — wide-character formatted input of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vfwscanf(FILE *restrict stream, const wchar_t *restrict format,
    va_list arg);
int vswscanf(const wchar_t *restrict ws, const wchar_t *restrict format,
    va_list arg);
int vwscanf(const wchar_t *restrict format, va_list arg);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *vfwscanf*( ), *vswscanf*( ), and *vwscanf*( ) functions shall be equivalent to the *fwscanf*( ), *swscanf*( ), and *wscanf*( ) functions, respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined in the **<stdarg.h>** header. These functions shall not invoke the *va_end* macro. As these functions invoke the *va_arg* macro, the value of *ap* after the return is unspecified.

RETURN VALUE

Refer to *fwscanf*( ).

ERRORS

Refer to *fwscanf*( ).

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions should call *va_end*(*ap*) afterwards to clean up.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 521), *fwscanf*( )

XBD **<stdarg.h>**, **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0706 [14] is applied.

76003 **NAME**

76004 vprintf — format the output of a stdarg argument list

76005 **SYNOPSIS**

76006 `#include <stdarg.h>`

76007 `#include <stdio.h>`

76008 `int vprintf(const char *restrict format, va_list ap);`

76009 **DESCRIPTION**

76010 Refer to *vfprintf( )*.

NAME

vscanf — format input of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>

int vscanf(const char *restrict format, va_list arg);
```

DESCRIPTION

Refer to *vfscanf( )*.

NAME

vsnprintf, vsprintf — format output of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>

int vsnprintf(char *restrict s, size_t n,
    const char *restrict format, va_list ap);
int vsprintf(char *restrict s, const char *restrict format,
    va_list ap);
```

DESCRIPTION

Refer to *vfprintf( )*.

76030 **NAME**

76031 vsscanf — format input of a stdarg argument list

76032 **SYNOPSIS**

76033 ```
#include <stdarg.h>
```

76034 ```
#include <stdio.h>
```

76035 ```
int vsscanf(const char *restrict s, const char *restrict format,
```

76036 ```
    va_list arg);
```

76037 **DESCRIPTION**

76038 Refer to *vfscanf( )*.

NAME

vswprintf — wide-character formatted output of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vswprintf(wchar_t *restrict ws, size_t n,
    const wchar_t *restrict format, va_list arg);
```

DESCRIPTION

Refer to *vfwprintf( )*.

NAME

vswscanf — wide-character formatted input of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vswscanf(const wchar_t *restrict ws, const wchar_t *restrict format,
    va_list arg);
```

DESCRIPTION

Refer to *vfwscanf( )*.

76059 **NAME**

76060 vwprintf — wide-character formatted output of a stdarg argument list

76061 **SYNOPSIS**

76062 `#include <stdarg.h>`
76063 `#include <stdio.h>`
76064 `#include <wchar.h>`

76065 `int vwprintf(const wchar_t *restrict format, va_list arg);`

76066 **DESCRIPTION**

76067 Refer to *vfwprintf( )*.

NAME

vwscanf — wide-character formatted input of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vwscanf(const wchar_t *restrict format, va_list arg);
```

DESCRIPTION

Refer to *vfwscanf( )*.

NAME

wait, waitpid — wait for a child process to stop or terminate

SYNOPSIS

```
#include <sys/wait.h>

pid_t wait(int *stat_loc);
pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

DESCRIPTION

The *wait*( ) and *waitpid*( ) functions shall obtain status information (see Section 2.12, on page 563) pertaining to one of the caller's child processes. The *wait*( ) function obtains status information for process termination from any child process. The *waitpid*( ) function obtains status information for process termination, and optionally process stop and/or continue, from a specified subset of the child processes.

The *wait*( ) function shall cause the calling thread to become blocked until status information generated by child process termination is made available to the thread, or until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process, or an error occurs. If termination status information is available prior to the call to *wait*( ), return shall be immediate. If termination status information is available for two or more child processes, the order in which their status is reported is unspecified.

As described in Section 2.12 (on page 563), the *wait*( ) and *waitpid*( ) functions consume the status information they obtain.

The behavior when multiple threads are blocked in *wait*( ), *waitid*( ), or *waitpid*( ) is described in Section 2.12 (on page 563).

The *waitpid*( ) function shall be equivalent to *wait*( ) if the *pid* argument is (**pid_t**)–1 and the *options* argument is 0. Otherwise, its behavior shall be modified by the values of the *pid* and *options* arguments.

The *pid* argument specifies a set of child processes for which *status* is requested. The *waitpid*( ) function shall only return the status of a child process from this set:

- If *pid* is equal to (**pid_t**)–1, *status* is requested for any child process. In this respect, *waitpid*( ) is then equivalent to *wait*( ).
- If *pid* is greater than 0, it specifies the process ID of a single child process for which *status* is requested.
- If *pid* is 0, *status* is requested for any child process whose process group ID is equal to that of the calling process.
- If *pid* is less than (**pid_t**)–1, *status* is requested for any child process whose process group ID is equal to the absolute value of *pid*.

The *options* argument is constructed from the bitwise-inclusive OR of zero or more of the following flags, defined in the **<sys/wait.h>** header:

XSI

WCONTINUED — The *waitpid*( ) function shall report the status of any continued child process specified by *pid* whose status has not been reported since it continued from a job control stop.

WNOHANG — The *waitpid*( ) function shall not suspend execution of the calling thread if *status* is not immediately available for one of the child processes specified by *pid*.

WUNTRACED The status of any child processes specified by *pid* that are stopped, and whose status has not yet been reported since they stopped, shall also be reported to the requesting process.

If *wait*( ) or *waitpid*( ) return because the status of a child process is available, these functions shall return a value equal to the process ID of the child process. In this case, if the value of the argument *stat_loc* is not a null pointer, information shall be stored in the location pointed to by *stat_loc*. The value stored at the location pointed to by *stat_loc* shall be 0 if and only if the status returned is from a terminated child process that terminated by one of the following means:

1. The process returned 0 from *main*( ).
2. The process called *_exit*( ) or *exit*( ) with a *status* argument of 0.
3. The process was terminated because the last thread in the process terminated.

Regardless of its value, this information may be interpreted using the following macros, which are defined in **<sys/wait.h>** and evaluate to integral expressions; the *stat_val* argument is the integer value pointed to by *stat_loc*.

WIFEXITED(*stat_val*)
Evaluates to a non-zero value if *status* was returned for a child process that terminated normally.

WEXITSTATUS(*stat_val*)
If the value of WIFEXITED(*stat_val*) is non-zero, this macro shall evaluate to the low-order 8 bits of the *status* argument that the child process passed to *_exit*( ) or *exit*( ), or the value the child process returned from *main*( ).

WIFSIGNALED(*stat_val*)
Evaluates to a non-zero value if *status* was returned for a child process that terminated due to the receipt of a signal that was not caught (see **<signal.h>**).

WCOREDUMP(*stat_val*)
If the value of WIFSIGNALED(*stat_val*) is non-zero, this macro shall evaluate to a non-zero value if the creation of a core image of the terminated child was attempted.

WTERMSIG(*stat_val*)
If the value of WIFSIGNALED(*stat_val*) is non-zero, this macro shall evaluate to the number of the signal that caused the termination of the child process.

WIFSTOPPED(*stat_val*)
Evaluates to a non-zero value if *status* was returned for a child process that stopped due to the receipt of a signal that was not caught (see **<signal.h>**).

WSTOPSIG(*stat_val*)
If the value of WIFSTOPPED(*stat_val*) is non-zero, this macro shall evaluate to the number of the signal that caused the child process to stop.

XSI WIFCONTINUED(*stat_val*)
Evaluates to a non-zero value if *status* was returned for a child process that has continued from a job control stop.

SPN It is unspecified whether the *status* value returned by calls to *wait*( ) or *waitpid*( ) for processes created by *posix_spawn*( ) or *posix_spawnp*( ) can indicate a WIFSTOPPED(*stat_val*) before subsequent calls to *wait*( ) or *waitpid*( ) indicate WIFEXITED(*stat_val*) as the result of an error detected before the new process image starts executing.

It is unspecified whether the *status* value returned by calls to *wait*( ) or *waitpid*( ) for processes

created by *posix_spawn*( ) or *posix_spawnp*( ) can indicate a WIFSIGNALED(*stat_val*) if a signal is sent to the parent's process group after *posix_spawn*( ) or *posix_spawnp*( ) is called.

If the information pointed to by *stat_loc* was stored by a call to *waitpid*( ) that specified the WUNTRACED flag and did not specify the WCONTINUED flag, exactly one of the macros WIFEXITED(**stat_loc*), WIFSIGNALED(**stat_loc*), and WIFSTOPPED(**stat_loc*) shall evaluate to a non-zero value.

XSI If the information pointed to by *stat_loc* was stored by a call to *waitpid*( ) that specified the WUNTRACED and WCONTINUED flags, exactly one of the macros WIFEXITED(**stat_loc*), WIFSIGNALED(**stat_loc*), WIFSTOPPED(**stat_loc*), and WIFCONTINUED(**stat_loc*) shall evaluate to a non-zero value.

If the information pointed to by *stat_loc* was stored by a call to *waitpid*( ) that did not specify the WUNTRACED or WCONTINUED flags, or by a call to the *wait*( ) function, exactly one of the macros WIFEXITED(**stat_loc*) and WIFSIGNALED(**stat_loc*) shall evaluate to a non-zero value.

XSI If the information pointed to by *stat_loc* was stored by a call to *waitpid*( ) that did not specify the WUNTRACED flag and specified the WCONTINUED flag, exactly one of the macros WIFEXITED(**stat_loc*), WIFSIGNALED(**stat_loc*), and WIFCONTINUED(**stat_loc*) shall evaluate to a non-zero value.

If the implementation queues the SIGCHLD signal, then if *wait*( ) or *waitpid*( ) returns because the status of a child process is available, any pending SIGCHLD signal associated with the process ID of the child process shall be discarded. Any other pending SIGCHLD signals shall remain pending.

Otherwise, if SIGCHLD is blocked, if *wait*( ) or *waitpid*( ) return because the status of a child process is available, any pending SIGCHLD signal shall be cleared unless the status of another child process is available.

For all other conditions, it is unspecified whether child *status* will be available when a SIGCHLD signal is delivered.

There may be additional implementation-defined circumstances under which *wait*( ) or *waitpid*( ) report *status*. This shall not occur unless the calling process or one of its child processes explicitly makes use of a non-standard extension. In these cases the interpretation of the reported *status* is implementation-defined.

If a parent process terminates without waiting for all of its child processes to terminate, the remaining child processes shall be assigned a new parent process ID corresponding to an implementation-defined system process.

RETURN VALUE

If *wait*( ) or *waitpid*( ) returns because the status of a child process is available, these functions shall return a value equal to the process ID of the child process for which *status* is reported. If *wait*( ) or *waitpid*( ) returns due to the delivery of a signal to the calling process, −1 shall be returned and *errno* set to [EINTR]. If *waitpid*( ) was invoked with WNOHANG set in *options*, it has at least one child process specified by *pid* for which *status* is not available, and *status* is not available for any process specified by *pid*, 0 is returned. Otherwise, −1 shall be returned, and *errno* set to indicate the error.

ERRORS

The *wait*( ) function shall fail if:

[ECHILD] The calling process has no existing unwaited-for child processes.

[EINTR] The function was interrupted by a signal. The value of the location pointed to by *stat_loc* is undefined.

The *waitpid*( ) function shall fail if:

[ECHILD] The process specified by *pid* does not exist or is not a child of the calling process, or the process group specified by *pid* does not exist or does not have any member process that is a child of the calling process.

[EINTR] The function was interrupted by a signal. The value of the location pointed to by *stat_loc* is undefined.

[EINVAL] The *options* argument is not valid.

EXAMPLES

Waiting for a Child Process and then Checking its Status

The following example demonstrates the use of *waitpid*( ), *fork*( ), and the macros used to interpret the status value returned by *waitpid*( ) (and *wait*( )). The code segment creates a child process which does some unspecified work. Meanwhile the parent loops performing calls to *waitpid*( ) to monitor the status of the child. The loop terminates when child termination is detected.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
...

pid_t child_pid, wpid;
int status;
child_pid = fork();
if (child_pid == -1) {      /* fork() failed */
    perror("fork");
    exit(EXIT_FAILURE);
}

if (child_pid == 0) {       /* This is the child */
    /* Child does some work and then terminates */
    ...

} else {                    /* This is the parent */
    do {
        wpid = waitpid(child_pid, &status, WUNTRACED
#ifdef WCONTINUED       /* Not all implementations support this */
        | WCONTINUED
#endif
        );
        if (wpid == -1) {
            perror("waitpid");
            exit(EXIT_FAILURE);
        }

        if (WIFEXITED(status)) {
            printf("child exited, status=%d\n", WEXITSTATUS(status));

        } else if (WIFSIGNALED(status)) {
```



```
                printf("child killed (signal %d)\n", WTERMSIG(status));
            } else if (WIFSTOPPED(status)) {
                printf("child stopped (signal %d)\n", WSTOPSIG(status));
#ifdef WIFCONTINUED     /* Not all implementations support this */
            } else if (WIFCONTINUED(status)) {
                printf("child continued\n");
#endif
            } else {    /* Non-standard case -- may never happen */
                printf("Unexpected status (0x%x)\n", status);
            }
        } while (!WIFEXITED(status) && !WIFSIGNALED(status));
}
```

Waiting for a Child Process in a Signal Handler for SIGCHLD

The following example demonstrates how to use *waitpid*() in a signal handler for SIGCHLD without passing –1 as the *pid* argument. (See the APPLICATION USAGE section below for the reasons why passing a *pid* of –1 is not recommended.) The method used here relies on the standard behavior of *waitpid*() when SIGCHLD is blocked. On historical non-conforming systems, the status of some child processes might not be reported.

```
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

#define CHILDREN 10

static void
handle_sigchld(int signum, siginfo_t *sinfo, void *unused)
{
    int sav_errno = errno;
    int status;

    /*
     * Obtain status information for the child which
     * caused the SIGCHLD signal and write its exit code
     * to stdout.
     */
    if (sinfo->si_code != CLD_EXITED)
    {
        static char msg[] = "wrong si_code\n";
        write(2, msg, sizeof msg - 1);
    }
    else if (waitpid(sinfo->si_pid, &status, 0) == -1)
    {
        static char msg[] = "waitpid() failed\n";
        write(2, msg, sizeof msg - 1);
    }
    else if (!WIFEXITED(status))
    {
```



```
76300                static char msg[] = "WIFEXITED was false\n";
76301                write(2, msg, sizeof msg - 1);
76302            }
76303            else
76304            {
76305                int code = WEXITSTATUS(status);
76306                char buf[2];
76307                buf[0] = '0' + code;
76308                buf[1] = '\n';
76309                write(1, buf, 2);
76310            }
76311            errno = sav_errno;
76312        }
76313        int
76314        main(void)
76315        {
76316            int i;
76317            pid_t pid;
76318            struct sigaction sa;
76319            sa.sa_flags = SA_SIGINFO;
76320            sa.sa_sigaction = handle_sigchld;
76321            sigemptyset(&sa.sa_mask);
76322            if (sigaction(SIGCHLD, &sa, NULL) == -1)
76323            {
76324                perror("sigaction");
76325                exit(EXIT_FAILURE);
76326            }
76327            for (i = 0; i < CHILDREN; i++)
76328            {
76329                switch (pid = fork())
76330                {
76331                case -1:
76332                    perror("fork");
76333                    exit(EXIT_FAILURE);
76334                case 0:
76335                    sleep(2);
76336                    _exit(i);
76337                }
76338            }
76339            /* Wait for all the SIGCHLD signals, then terminate on SIGALRM */
76340            alarm(3);
76341            for (;;)
76342                pause();
76343            return 0; /* NOTREACHED */
76344        }
```


APPLICATION USAGE

Calls to *wait*( ) will collect information about any child process. This may result in interactions with other interfaces that may be waiting for their own children (such as by use of *system*( )). For this and other reasons it is recommended that portable applications not use *wait*( ), but instead use *waitpid*( ). For these same reasons, the use of *waitpid*( ) with a *pid* argument of –1, and the use of *waitid*( ) with the *idtype* argument set to P_ALL, are also not recommended for portable applications.

XSI As specified in Consequences of Process Termination (on page 568), if the calling process has SA_NOCLDWAIT set or has SIGCHLD set to SIG_IGN, then the termination of a child process will not cause status information to become available to a thread blocked in *wait*( ), *waitid*( ), or *waitpid*( ). Thus, a thread blocked in one of the wait functions will remain blocked unless some other condition causes the thread to resume execution (such as an [ECHILD] failure due to no remaining children in the set of waited-for children).

RATIONALE

A call to the *wait*( ) or *waitpid*( ) function only returns *status* on an immediate child process of the calling process; that is, a child that was produced by a single *fork*( ) call (perhaps followed by an *exec* or other function calls) from the parent. If a child produces grandchildren by further use of *fork*( ), none of those grandchildren nor any of their descendants affect the behavior of a *wait*( ) from the original parent process. Nothing in this volume of POSIX.1-2024 prevents an implementation from providing extensions that permit a process to get *status* from a grandchild or any other process, but a process that does not use such extensions must be guaranteed to see *status* from only its direct children.

The *waitpid*( ) function is provided for three reasons:

1. To support job control
2. To permit a non-blocking version of the *wait*( ) function
3. To permit a library routine, such as *system*( ) or *pclose*( ), to wait for its children without interfering with other terminated children for which the process has not waited

The first two of these facilities are based on the *wait3*( ) function provided by 4.3 BSD. The function uses the *options* argument, which is equivalent to an argument to *wait3*( ). The WUNTRACED flag is used only in conjunction with job control on systems supporting job control. Its name comes from 4.3 BSD and refers to the fact that there are two types of stopped processes in that implementation: processes being traced via the *ptrace*( ) debugging facility and (untraced) processes stopped by job control signals. Since *ptrace*( ) is not part of this volume of POSIX.1-2024, only the second type is relevant. The name WUNTRACED was retained because its usage is the same, even though the name is not intuitively meaningful in this context.

The third reason for the *waitpid*( ) function is to permit independent sections of a process to spawn and wait for children without interfering with each other. For example, the following problem occurs in developing a portable shell, or command interpreter:

```
stream = popen("/bin/true");
(void) system("sleep 100");
(void) pclose(stream);
```

On all historical implementations, the final *pclose*( ) fails to reap the *wait*( ) *status* of the *popen*( ).

The status values are retrieved by macros, rather than given as specific bit encodings as they are in most historical implementations (and thus expected by existing programs). This was necessary to eliminate a limitation on the number of signals an implementation can support that was inherent in the traditional encodings. This volume of POSIX.1-2024 does require that a *status*

value of zero corresponds to a process calling *_exit*(0), as this is the most common encoding expected by existing programs. Some of the macro names were adopted from 4.3 BSD.

These macros syntactically operate on an arbitrary integer value. The behavior is undefined unless that value is one stored by a successful call to *wait*( ) or *waitpid*( ) in the location pointed to by the *stat_loc* argument. An early proposal attempted to make this clearer by specifying each argument as **stat_loc* rather than *stat_val*. However, that did not follow the conventions of other specifications in this volume of POSIX.1-2024 or traditional usage. It also could have implied that the argument to the macro must literally be **stat_loc*; in fact, that value can be stored or passed as an argument to other functions before being interpreted by these macros.

The extension that affects *wait*( ) and *waitpid*( ) and is common in historical implementations is the *ptrace*( ) function. It is called by a child process and causes that child to stop and return a *status* that appears identical to the *status* indicated by WIFSTOPPED. The *status* of *ptrace*( ) children is traditionally returned regardless of the WUNTRACED flag (or by the *wait*( ) function). Most applications do not need to concern themselves with such extensions because they have control over what extensions they or their children use. However, applications, such as command interpreters, that invoke arbitrary processes may see this behavior when those arbitrary processes misuse such extensions.

On implementations that support the creation of a file containing a core image on some process terminations, the WCOREDUMP(*stat_val*) macro indicates whether creation of a core image was attempted. If it returns a non-zero value this does not necessarily mean that the core image was created, only that it was attempted. For example, if the RLIMIT_CORE limit for the process is 0, this prevents creation of the file; WCOREDUMP(*stat_val*) returning non-zero in this case indicates that the file would have been created if the limit had not been 0.

Allowing the *wait*( ) family of functions to discard a pending SIGCHLD signal that is associated with a successfully waited-for child process puts them into the *sigwait*( ) and *sigwaitinfo*( ) category with respect to SIGCHLD.

This definition allows implementations to treat a pending SIGCHLD signal as accepted by the process in *wait*( ), with the same meaning of ``accepted'' as when that word is applied to the *sigwait*( ) family of functions.

Allowing the *wait*( ) family of functions to behave this way permits an implementation to be able to deal precisely with SIGCHLD signals.

In particular, an implementation that does accept (discard) the SIGCHLD signal can make the following guarantees regardless of the queuing depth of signals in general (the list of waitable children can hold the SIGCHLD queue):

1. If a SIGCHLD signal handler is established via *sigaction*( ) without the SA_RESETHAND flag, SIGCHLD signals can be accurately counted; that is, exactly one SIGCHLD signal will be delivered to or accepted by the process for every child process that terminates.
2. A single *wait*( ) issued from a SIGCHLD signal handler can be guaranteed to return immediately with status information for a child process.
3. When SA_SIGINFO is requested, the SIGCHLD signal handler can be guaranteed to receive a non-null pointer to a **siginfo_t** structure that describes a child process for which a wait via *waitpid*( ) or *waitid*( ) will not block or fail.
4. The *system*( ) function will not cause the SIGCHLD handler of a process to be called as a result of the *fork*( )/*exec* executed within *system*( ) because *system*( ) will accept the SIGCHLD signal when it performs a *waitpid*( ) for its child process. This is a desirable behavior of *system*( ) so that it can be used in a library without causing side-effects to the

application linked with the library.

An implementation that does not permit the *wait*( ) family of functions to accept (discard) a pending SIGCHLD signal associated with a successfully waited-for child, cannot make the guarantees described above for the following reasons:

Guarantee #1
: Although it might be assumed that reliable queuing of all SIGCHLD signals generated by the system can make this guarantee, the counter-example is the case of a process that blocks SIGCHLD and performs an indefinite loop of *fork*( )/*wait*( ) operations. If the implementation supports queued signals, then eventually the system will run out of memory for the queue. The guarantee cannot be made because there must be some limit to the depth of queuing.

Guarantees #2 and #3
: These cannot be guaranteed unless the *wait*( ) family of functions accepts the SIGCHLD signal. Otherwise, a *fork*( )/*wait*( ) executed while SIGCHLD is blocked (as in the *system*( ) function) will result in an invocation of the handler when SIGCHLD is unblocked, after the process has disappeared.

Guarantee #4
: Although possible to make this guarantee, *system*( ) would have to set the SIGCHLD handler to SIG_DFL so that the SIGCHLD signal generated by its *fork*( ) would be discarded (the SIGCHLD default action is to be ignored), then restore it to its previous setting. This would have the undesirable side-effect of discarding all SIGCHLD signals pending to the process.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.12 (on page 563), *exec*, *exit*( ), *fork*( ), *system*( ), *waitid*( )

XBD Section 4.15.2 (on page 104), **<signal.h>**, **<sys/wait.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The processing of the SIGCHLD signal and the [ECHILD] error is clarified.

The semantics of WIFSTOPPED(*stat_val*), WIFEXITED(*stat_val*), and WIFSIGNALED(*stat_val*) are defined with respect to *posix_spawn*( ) or *posix_spawnp*( ) for alignment with IEEE Std 1003.1d-1999.

The DESCRIPTION is updated for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/145 is applied, adding the example to the

EXAMPLES section.

Issue 7

SD5-XSH-ERN-202 is applied.

APPLICATION USAGE is added, recommending that the *wait*( ) function not be used.

An additional example for *waitpid*( ) is added.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0707 [421], XSH/TC1-2008/0708 [166], XSH/TC1-2008/0709 [166], and XSH/TC1-2008/0710 [69] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0384 [690], XSH/TC2-2008/0385 [691], and XSH/TC2-2008/0386 [690] are applied.

Issue 8

Austin Group Defect 1116 is applied, removing text related to the Realtime Signals Extension option that existed in earlier versions of this standard.

Austin Group Defects 1141 and 1363 are applied, adding WCOREDUMP, changing the description of WIFSTOPPED, and changing the RATIONALE section.

Austin Group Defect 1570 is applied, removing extra spacing in `"=="`.

NAME

waitid — wait for a child process to change state

SYNOPSIS

```
#include <sys/wait.h>

int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
```

DESCRIPTION

The *waitid*( ) function shall obtain status information (see Section 2.12, on page 563) pertaining to termination, stop, and/or continue events in one of the caller's child processes.

The *waitid*( ) function shall cause the calling thread to become blocked until an error occurs or status information becomes available to the calling thread that satisfies all of the following properties (``matching status information''):

- The status information is from one of the child processes in the set of child processes specified by the *idtype* and *id* arguments.
- The state change in the status information matches one of the state change flags set in the *options* argument.

If matching status information is available prior to the call to *waitid*( ), return shall be immediate. If matching status information is available for two or more child processes, the order in which their status is reported is unspecified.

As described in Section 2.12 (on page 563), the *waitid*( ) function consumes the status information it obtains unless the WNOWAIT flag is set in the *options* argument.

The behavior when multiple threads are blocked in *wait*( ), *waitid*( ), or *waitpid*( ) is described in Section 2.12 (on page 563).

The *waitid*( ) function shall record the obtained status information in the structure pointed to by *infop*. The fields of the structure pointed to by *infop* shall be filled in as described under ``Pointer to a Function'' in Section 2.4.3 (on page 516).

The *idtype* and *id* arguments are used to specify which children *waitid*( ) waits for.

If *idtype* is P_PID, *waitid*( ) shall wait for the child with a process ID equal to (**pid_t**)*id*.

If *idtype* is P_PGID, *waitid*( ) shall wait for any child with a process group ID equal to (**pid_t**)*id*.

If *idtype* is P_ALL, *waitid*( ) shall wait for any children and *id* is ignored.

The *options* argument is used to specify which state changes *waitid*( ) shall wait for. It is formed by OR'ing together the following flags:

WCONTINUED Status shall be returned for any continued child process whose status either has not been reported since it continued from a job control stop or has been reported only by calls to *waitid*( ) with the WNOWAIT flag set.

WEXITED Wait for processes that have terminated.

WNOHANG Do not hang if no status is available; return immediately.

WNOWAIT Keep the process whose status is returned in *infop* in a waitable state. This shall not affect the state of the process; the process may be waited for again after this call completes.

WSTOPPED Status shall be returned for any child that has stopped upon receipt of a signal, and whose status either has not been reported since it stopped or has been reported only by calls to *waitid*( ) with the WNOWAIT flag set.

Applications shall specify at least one of the flags WEXITED, WSTOPPED, or WCONTINUED to be OR'ed in with the *options* argument.

The application shall ensure that the *infop* argument points to a **siginfo_t** structure. If *waitid*( ) returns because a child process was found that satisfied the conditions indicated by the arguments *idtype* and *options*, then the structure pointed to by *infop* shall be filled in by the system with the status of the process; the *si_signo* member shall be set equal to SIGCHLD. If *waitid*( ) returns because WNOHANG was specified and status is not available for any process specified by *idtype* and *id*, then the *si_signo* and *si_pid* members of the structure pointed to by *infop* shall be set to zero and the values of other members of the structure are unspecified.

RETURN VALUE

If WNOHANG was specified and status is not available for any process specified by *idtype* and *id*, 0 shall be returned. If *waitid*( ) returns due to the change of state of one of its children, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *waitid*( ) function shall fail if:

[ECHILD] The calling process has no existing unwaited-for child processes.

[EINTR] The *waitid*( ) function was interrupted by a signal. The values of the fields of the structure pointed to by *infop* are undefined.

[EINVAL] An invalid value was specified for *options*, or *idtype* and *id* specify an invalid set of processes.

EXAMPLES

None.

APPLICATION USAGE

Calls to *waitid*( ) with *idtype* equal to P_ALL will collect information about any child process. This may result in interactions with other interfaces that may be waiting for their own children (such as by use of *system*( )). For this reason it is recommended that portable applications not use *waitid*( ) with idtype of P_ALL. See also APPLICATION USAGE for *wait*( ).

XSI As specified in Consequences of Process Termination (on page 568), if the calling process has SA_NOCLDWAIT set or has SIGCHLD set to SIG_IGN, then the termination of a child process will not cause status information to become available to a thread blocked in *wait*( ), *waitid*( ), or *waitpid*( ). Thus, a thread blocked in one of the wait functions will remain blocked unless some other condition causes the thread to resume execution (such as an [ECHILD] failure due to no remaining children in the set of waited-for children).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4.3 (on page 516), Section 2.12 (on page 563), *exec*, *exit*( ), *wait*( )

XBD **<signal.h>**, **<sys/wait.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #060 is applied, updating the DESCRIPTION.

The *waitid*( ) function is moved from the XSI option to the Base.

APPLICATION USAGE is added, recommending that the *waitid*( ) function not be used with *idtype* equal to P_ALL.

The description of the WNOHANG flag is updated.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0711 [154], XSH/TC1-2008/0712 [154], and XSH/TC1-2008/0713 [153] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0387 [690] is applied.

Issue 8

Austin Group Defect 1332 is applied, changing the description of WEXITED.

Austin Group Defect 1547 is applied, changing the description of [EINTR].

76597 **NAME**

76598 waitpid — wait for a child process to stop or terminate

76599 **SYNOPSIS**

76600 `#include <sys/wait.h>`

76601 `pid_t waitpid(pid_t pid, int *stat_loc, int options);`

76602 **DESCRIPTION**

76603 Refer to *wait( )*.

NAME

wcpcpy — copy a wide-character string, returning a pointer to its end

SYNOPSIS

CX
```
#include <wchar.h>

wchar_t *wcpcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);
```

DESCRIPTION

Refer to *wcscpy( )*.

NAME

wcpncpy — copy a fixed-size wide-character string, returning a pointer to its end

SYNOPSIS

CX
```
#include <wchar.h>

wchar_t *wcpncpy(wchar_t restrict *ws1, const wchar_t *restrict ws2,
    size_t n);
```

DESCRIPTION

Refer to *wcsncpy( )*.

NAME

wcrtomb — convert a wide-character code to a character (restartable)

SYNOPSIS

```
#include <wchar.h>

size_t wcrtomb(char *restrict s, wchar_t wc, mbstate_t *restrict ps);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

If *s* is a null pointer, the *wcrtomb*( ) function shall be equivalent to the call:

```
wcrtomb(buf, L'\0', ps)
```

where *buf* is an internal buffer.

If *s* is not a null pointer, the *wcrtomb*( ) function shall determine the number of bytes needed to represent the character that corresponds to the wide character given by *wc* (including any shift sequences), and store the resulting bytes in the array whose first element is pointed to by *s*. At most {MB_CUR_MAX} bytes are stored. If *wc* is a null wide character, a null byte shall be stored, preceded by any shift sequence needed to restore the initial shift state. The resulting state described shall be the initial conversion state.

If *ps* is a null pointer, the *wcrtomb*( ) function shall use its own internal **mbstate_t** object, which is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence. The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls *wcrtomb*( ).

If called with a null *ps* argument, the *wcrtomb*( ) function need not be thread-safe; however, such calls shall avoid data races with calls to *wcrtomb*( ) with a non-null argument and with calls to all other functions.

The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

The *wcrtomb*( ) function shall not change the setting of *errno* if successful.

RETURN VALUE

The *wcrtomb*( ) function shall return the number of bytes stored in the array object (including any shift sequences). When *wc* is not a valid wide character, an encoding error shall occur. In this case, the function shall store the value of the macro [EILSEQ] in *errno* and shall return (**size_t**)–1; the conversion state shall be undefined.

ERRORS

The *wcrtomb*( ) function shall fail if:

[EILSEQ] An invalid wide-character code is detected.

The *wcrtomb*( ) function may fail if:

CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mbsinit( ), *wcsrtombs*( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

In the DESCRIPTION, a note on using this function in a threaded application is added.

Extensions beyond the ISO C standard are marked.

The normative text is updated to avoid use of the term ``must'' for application requirements.

The *wcrtomb*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *wcrtomb*( ) function need not be thread-safe if called with a NULL *ps* argument.

Austin Group Interpretation 1003.1-2001 #170 is applied.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0714 [88] and XSH/TC1-2008/0715 [105] are applied.

Issue 8

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018 standard.

NAME

wcscasecmp, wcscasecmp_l, wcsncasecmp, wcsncasecmp_l — case-insensitive wide-character string comparison

SYNOPSIS

CX
```
#include <wchar.h>

int wcscasecmp(const wchar_t *ws1, const wchar_t *ws2);
int wcscasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
    locale_t locale);
int wcsncasecmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
int wcsncasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
    size_t n, locale_t locale);
```

DESCRIPTION

The *wcscasecmp*( ) and *wcsncasecmp*( ) functions are the wide-character equivalent of the *strcasecmp*( ) and *strncasecmp*( ) functions, respectively.

The *wcscasecmp*( ) and *wcscasecmp_l*( ) functions shall compare, while ignoring differences in case, the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*.

The *wcsncasecmp*( ) and *wcsncasecmp_l*( ) functions shall compare, while ignoring differences in case, not more than *n* wide-characters from the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*.

The *wcscasecmp*( ) and *wcsncasecmp*( ) functions use the current locale to determine the case of the wide characters.

The *wcscasecmp_l*( ) and *wcsncasecmp_l*( ) functions use the locale represented by *locale* to determine the case of the wide characters.

When the *LC_CTYPE* category of the locale being used is from the POSIX locale, these functions shall behave as if the wide-character strings had been converted to lowercase and then a comparison of wide-character codes performed. Otherwise, the results are unspecified.

The information for *wcscasecmp_l*( ) and *wcsncasecmp_l*( ) about the case of the characters comes from the locale represented by *locale*.

The behavior is undefined if the *locale* argument to *wcscasecmp_l*( ) or *wcsncasecmp_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

Upon completion, the *wcscasecmp*( ) and *wcscasecmp_l*( ) functions shall return an integer greater than, equal to, or less than 0 if the wide-character string pointed to by *ws1* is, ignoring case, greater than, equal to, or less than the wide-character string pointed to by *ws2*, respectively.

Upon completion, the *wcsncasecmp*( ) and *wcsncasecmp_l*( ) functions shall return an integer greater than, equal to, or less than 0 if the possibly null wide-character terminated string pointed to by *ws1* is, ignoring case, greater than, equal to, or less than the possibly null wide-character terminated string pointed to by *ws2*, respectively.

No return values are reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcasecmp( ), *wcscmp*( ), *wcsncmp*( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 7.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0716 [294], XSH/TC1-2008/0717 [283], and XSH/TC1-2008/0718 [283] are applied.

NAME

wcscat — concatenate two wide-character strings

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *restrict ws1, const wchar_t *restrict ws2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *wcscat*( ) function shall append a copy of the wide-character string pointed to by *ws2* (including the terminating null wide-character code) to the end of the wide-character string pointed to by *ws1*. The initial wide-character code of *ws2* shall overwrite the null wide-character code at the end of *ws1*. If copying takes place between objects that overlap, the behavior is undefined.

CX The *wcscat*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

The *wcscat*( ) function shall return *ws1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcsncat( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 6

The Open Group Corrigendum U040/2 is applied. In the RETURN VALUE section, *s1* is changed to *ws1*.

The *wcscat*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *wcscat*( ) does not change the setting of *errno* on valid input.

NAME

wcschr — wide-character string scanning operation

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcschr(const wchar_t *ws, wchar_t wc);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *wcschr*( ) function shall locate the first occurrence of *wc* in the wide-character string pointed to by *ws*. The application shall ensure that the value of *wc* is a character representable as a type **wchar_t** and a wide-character code corresponding to a valid character in the current locale. The terminating null wide-character code is considered to be part of the wide-character string.

CX The *wcschr*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

Upon completion, *wcschr*( ) shall return a pointer to the wide-character code, or a null pointer if the wide-character code is not found.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcsrchr( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *wcschr*( ) does not change the setting of *errno* on valid input.

NAME

wcscmp — compare two wide-character strings

SYNOPSIS

```
#include <wchar.h>

int wcscmp(const wchar_t *ws1, const wchar_t *ws2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *wcscmp*( ) function shall compare the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*.

The sign of a non-zero return value shall be determined by the sign of the difference between the values of the first pair of wide-character codes that differ in the objects being compared.

CX The *wcscmp*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

Upon completion, *wcscmp*( ) shall return an integer greater than, equal to, or less than 0, if the wide-character string pointed to by *ws1* is greater than, equal to, or less than the wide-character string pointed to by *ws2*, respectively.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscasecmp( ), *wcsncmp*( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *wcscmp*( ) does not change the setting of *errno* on valid input.

NAME

wcscoll, wcscoll_l — wide-character string comparison using collating information

SYNOPSIS

```
#include <wchar.h>

int wcscoll(const wchar_t *ws1, const wchar_t *ws2);
int wcscoll_l(const wchar_t *ws1, const wchar_t *ws2,
    locale_t locale);
```

DESCRIPTION

For *wcscoll*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *wcscoll*( ) and *wcscoll_l*( ) functions shall compare the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*, both interpreted as appropriate to the *LC_COLLATE* category of the current locale, or the locale represented by *locale*, respectively.

The *wcscoll*( ) and *wcscoll_l*( ) functions shall not change the setting of *errno* if successful.

An application wishing to check for error situations should set *errno* to 0 before calling *wcscoll*( ) or *wcscoll_l*( ). If *errno* is non-zero on return, an error has occurred.

The behavior is undefined if the *locale* argument to *wcscoll_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

Upon successful completion, *wcscoll*( ) and *wcscoll_l*( ) shall return an integer greater than, equal to, or less than 0, according to whether the wide-character string pointed to by *ws1* is greater than, equal to, or less than the wide-character string pointed to by *ws2*, when both are interpreted as appropriate to the current locale, or to the locale represented by *locale*, respectively. On error, *wcscoll*( ) and *wcscoll_l*( ) shall set *errno*, but no return value is reserved to indicate an error.

ERRORS

These functions may fail if:

[EINVAL] The *ws1* or *ws2* arguments contain wide-character codes outside the domain of the collating sequence.

EXAMPLES

None.

APPLICATION USAGE

The *wcsxfrm*( ) and *wcscmp*( ) functions should be used for sorting large lists.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscmp( ), *wcsxfrm*( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

Issue 7

The *wcscoll_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0719 [302], XSH/TC1-2008/0720 [283], and XSH/TC1-2008/0721 [283] are applied.

NAME

wcpcpy, wcscpy — copy a wide-character string, returning a pointer to its end

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcpcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);
wchar_t *wcscpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);
```

DESCRIPTION

For *wcscpy*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *wcpcpy*( ) and *wcscpy*( ) functions shall copy the wide-character string pointed to by *ws2* (including the terminating null wide-character code) into the array pointed to by *ws1*.

The application shall ensure that there is room for at least *wcslen*(*ws2*)+1 wide characters in the *ws1* array, and that the *ws2* and *ws1* arrays do not overlap.

If copying takes place between objects that overlap, the behavior is undefined.

The *wcscpy*( ) and *wcpcpy*( ) functions shall not change the setting of *errno* on valid input.

RETURN VALUE

The *wcpcpy*( ) function shall return a pointer to the terminating null wide-character code copied into the *ws1* buffer.

The *wcscpy*( ) function shall return *ws1*.

No return values are reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcpy( ), *wcsdup*( ), *wcsncpy*( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 6

The *wcscpy*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

The *wcpcpy*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

76945 **Issue 8**

76946 Austin Group Defect 448 is applied, adding a requirement that *wcscpy*( ) and *wcpcpy*( ) do not
76947 change the setting of *errno* on valid input.

NAME

wcscspn — get the length of a complementary wide substring

SYNOPSIS

```
#include <wchar.h>

size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *wcscspn*( ) function shall compute the length (in wide characters) of the maximum initial segment of the wide-character string pointed to by *ws1* which consists entirely of wide-character codes *not* from the wide-character string pointed to by *ws2*.

CX The *wcscspn*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

The *wcscspn*( ) function shall return the length of the initial substring of *ws1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcsspn( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

The RETURN VALUE section is updated to indicate that *wcscspn*( ) returns the length of *ws1*, rather than *ws1* itself.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *wcscspn*( ) does not change the setting of *errno* on valid input.

NAME

wcsdup — duplicate a wide-character string

SYNOPSIS

CX
```
#include <wchar.h>

wchar_t *wcsdup(const wchar_t *string);
```

DESCRIPTION

The *wcsdup*( ) function is the wide-character equivalent of the *strdup*( ) function.

The *wcsdup*( ) function shall return a pointer to a new wide-character string, allocated as if by a call to *malloc*( ), which is the duplicate of the wide-character string *string*. The returned pointer can be passed to *free*( ). A null pointer is returned if the new wide-character string cannot be created.

RETURN VALUE

Upon successful completion, the *wcsdup*( ) function shall return a pointer to the newly allocated wide-character string. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *wcsdup*( ) function shall fail if:

[ENOMEM] Memory large enough for the duplicate string could not be allocated.

EXAMPLES

None.

APPLICATION USAGE

For functions that allocate memory as if by *malloc*( ), the application should release such memory when it is no longer required by a call to *free*( ). For *wcsdup*( ), this is the return value.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

free( ), *strdup*( ), *wcscpy*( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 7.

77016 **NAME**

77017 wcsftime — convert date and time to a wide-character string

77018 **SYNOPSIS**

77019
```
#include <wchar.h>
```

77020
```
size_t wcsftime(wchar_t *restrict wcs, size_t maxsize,
```
77021
```
    const wchar_t *restrict format, const struct tm *restrict timeptr);
```

77022 **DESCRIPTION**

77023 CX The functionality described on this reference page is aligned with the ISO C standard. Any
77024 conflict between the requirements described here and the ISO C standard is unintentional. This
77025 volume of POSIX.1-2024 defers to the ISO C standard.

77026 The *wcsftime*( ) function shall be equivalent to the *strftime*( ) function, except that:

77027 • The argument *wcs* points to the initial element of an array of wide characters into which
77028 the generated output is to be placed.

77029 • The argument *maxsize* indicates the maximum number of wide characters to be placed in
77030 the output array.

77031 • The argument *format* is a wide-character string and the conversion specifications are
77032 replaced by corresponding sequences of wide characters. It is unspecified whether an
77033 encoding error occurs if the format string contains **wchar_t** values that do not correspond
77034 to members of the character set of the current locale.

77035 CX • Field widths specify the number of wide characters instead of the number of bytes.

77036 • The return value indicates the number of wide characters placed in the output array.

77037 If copying takes place between objects that overlap, the behavior is undefined.

77038 **RETURN VALUE**

77039 If the total number of resulting wide-character codes including the terminating null wide-
77040 character code is no more than *maxsize*, *wcsftime*( ) shall return the number of wide-character
77041 codes placed into the array pointed to by *wcs*, not including the terminating null wide-character
77042 code. Otherwise, zero is returned and the contents of the array are unspecified.

77043 **ERRORS**

77044 No errors are defined.

77045 **EXAMPLES**

77046 None.

77047 **APPLICATION USAGE**

77048 None.

77049 **RATIONALE**

77050 None.

77051 **FUTURE DIRECTIONS**

77052 None.

77053 **SEE ALSO**

77054 *strftime*( )

77055 XBD **<wchar.h>**

77056 **CHANGE HISTORY**

77057 First released in Issue 4.

77058 **Issue 5**

77059 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

77060 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of the *format*
77061 argument is changed from **const char *** to **const wchar_t ***.

77062 **Issue 6**

77063 The *wcsftime*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

77064 **Issue 7**

77065 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0388 [73] and XSH/TC2-2008/0389
77066 [740] are applied.

77067 **NAME**

77068 wcslcat, wcslcpy — size-bounded wide string concatenation and copying

77069 **SYNOPSIS**

77070 CX
```
#include <wchar.h>

size_t wcslcat(wchar_t *restrict dst, const wchar_t *restrict src,
    size_t dstsize);
size_t wcslcpy(wchar_t *restrict dst, const wchar_t *restrict src,
    size_t dstsize);
```

77075 **DESCRIPTION**

77076 The *wcslcpy*( ) and *wcslcat*( ) functions copy and concatenate wide strings, stopping when either a
77077 terminating null wide-character code in the source wide string is encountered or the specified
77078 full size (in wide-character codes) of the destination buffer is reached. They null terminate the
77079 result if there is room. The application should ensure that room for the terminating null wide-
77080 character code is included in *dstsize*.

77081 The *wcslcpy*( ) function shall copy not more than *dstsize* – 1 wide-character codes from the wide
77082 string pointed to by *src* to the array pointed to by *dst*; a terminating null wide-character code in
77083 *src* and wide-character codes that follow it shall not be copied. A terminating null wide-
77084 character code shall be appended to the result, unless *dstsize* is 0. If copying takes place between
77085 objects that overlap, the behavior is undefined.

77086 The *wcslcat*( ) function shall append not more than *dstsize* – *wcslen*(*dst*) – 1 wide-character codes
77087 from the wide string pointed to by *src* to the end of the wide string pointed to by *dst*; a
77088 terminating null wide-character code in *src* and wide-character codes that follow it shall not be
77089 appended. The initial wide-character code of *src* shall overwrite the null wide-character code at
77090 the end of *dst*. A terminating null wide-character code shall be appended to the result, unless its
77091 location would be at or beyond *dst* + *dstsize*. If copying takes place between objects that overlap,
77092 the behavior is undefined.

77093 The *wcslcpy*( ) and *wcslcat*( ) functions shall not change the setting of *errno* on valid input.

77094 **RETURN VALUE**

77095 Upon successful completion, the *wcslcpy*( ) function shall return the length of the wide string
77096 pointed to by *src*; that is, the number of wide-character codes in the wide string, not including
77097 the terminating null wide-character code.

77098 Upon successful completion, the *wcslcat*( ) function shall return the initial length of the wide
77099 string pointed to by *dst* plus the length of the wide string pointed to by *src*.

77100 No return values are reserved to indicate an error.

77101 **ERRORS**

77102 No errors are defined.

77103 **EXAMPLES**

77104 None.

77105 **APPLICATION USAGE**

77106 The return value of the *wcslcpy*( ) and *wcslcat*( ) functions follows the same convention as
77107 *snprintf*( ); that is, they return the total length (in wide-character codes) of the wide string they
77108 tried to create. If the return value is greater than or equal to *dstsize*, the output wide string has
77109 been truncated.

77110 **RATIONALE**

77111 None.

77112 **FUTURE DIRECTIONS**

77113 None.

77114 **SEE ALSO**

77115 *fprintf*( ), *strlcat*( ), *wcslen*( ), *wcsncat*( ), *wcsncpy*( )

77116 XBD **<wchar.h>**

77117 **CHANGE HISTORY**

77118 First released in Issue 8.

77119 **NAME**

77120 wcslen, wcsnlen — get length of a fixed-sized wide-character string

77121 **SYNOPSIS**

77122
```
#include <wchar.h>
```

77123
```
size_t wcslen(const wchar_t *ws);
```

77124 CX
```
size_t wcsnlen(const wchar_t *ws, size_t maxlen);
```

77125 **DESCRIPTION**

77126 CX For *wcslen*( ): The functionality described on this reference page is aligned with the ISO C
77127 standard. Any conflict between the requirements described here and the ISO C standard is
77128 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

77129 The *wcslen*( ) function shall compute the number of wide-character codes in the wide-character
77130 string to which *ws* points, not including the terminating null wide-character code.

77131 CX The *wcsnlen*( ) function shall compute the smaller of the number of wide characters in the array
77132 to which *ws* points, not including any terminating null wide-character code, and the value of
77133 *maxlen*. The *wcsnlen*( ) function shall never examine more than the first *maxlen* characters of the
77134 wide-character array pointed to by *ws*.

77135 CX The *wcslen*( ) and *wcsnlen*( ) functions shall not change the setting of *errno* on valid input.

77136 **RETURN VALUE**

77137 The *wcslen*( ) function shall return the length of *ws*.

77138 CX The *wcsnlen*( ) function shall return the number of wide characters preceding the first null wide-
77139 character code in the array to which *ws* points, if *ws* contains a null wide-character code within
77140 the first *maxlen* wide characters; otherwise, it shall return *maxlen*.

77141 No return values are reserved to indicate an error.

77142 **ERRORS**

77143 No errors are defined.

77144 **EXAMPLES**

77145 None.

77146 **APPLICATION USAGE**

77147 None.

77148 **RATIONALE**

77149 None.

77150 **FUTURE DIRECTIONS**

77151 None.

77152 **SEE ALSO**

77153 *strlen*( ), *wcslcat*( )

77154 XBD **<wchar.h>**

77155 **CHANGE HISTORY**

77156 First released in Issue 4. Derived from the MSE working draft.

77157 **Issue 7**

77158 The *wcsnlen*( ) function is added from The Open Group Technical Standard, 2006, Extended API
77159 Set Part 1.

77160 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0390 [560] is applied.

77161 **Issue 8**

77162 Austin Group Defect 448 is applied, adding a requirement that *wcslen*( ) and *wcsnlen*( ) do not
77163 change the setting of *errno* on valid input.

77164 Austin Group Defect 986 is applied, adding *wcslcat*( ) to the SEE ALSO section.

77165 **NAME**

77166 wcsncasecmp, wcsncasecmp_l — case-insensitive wide-character string comparison

77167 **SYNOPSIS**

77168 CX
```
#include <wchar.h>

int wcsncasecmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
int wcsncasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
    size_t n, locale_t locale);
```

77172 **DESCRIPTION**

77173 Refer to *wcscasecmp( )*.

77174 **NAME**

77175 wcsncat — concatenate a wide-character string with part of another

77176 **SYNOPSIS**

77177
```
#include <wchar.h>
```

77178
```
wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict ws2,
```
77179
```
    size_t n);
```

77180 **DESCRIPTION**

77181 CX The functionality described on this reference page is aligned with the ISO C standard. Any
77182 conflict between the requirements described here and the ISO C standard is unintentional. This
77183 volume of POSIX.1-2024 defers to the ISO C standard.

77184 The *wcsncat*( ) function shall append not more than *n* wide-character codes (a null wide-
77185 character code and wide-character codes that follow it are not appended) from the array pointed
77186 to by *ws2* to the end of the wide-character string pointed to by *ws1*. The initial wide-character
77187 code of *ws2* shall overwrite the null wide-character code at the end of *ws1*. A terminating null
77188 wide-character code shall always be appended to the result. If copying takes place between
77189 objects that overlap, the behavior is undefined.

77190 CX The *wcsncat*( ) function shall not change the setting of *errno* on valid input.

77191 **RETURN VALUE**

77192 The *wcsncat*( ) function shall return *ws1*; no return value is reserved to indicate an error.

77193 **ERRORS**

77194 No errors are defined.

77195 **EXAMPLES**

77196 None.

77197 **APPLICATION USAGE**

77198 None.

77199 **RATIONALE**

77200 None.

77201 **FUTURE DIRECTIONS**

77202 None.

77203 **SEE ALSO**

77204 *wcscat*( ), *wcslcat*( )

77205 XBD **<wchar.h>**

77206 **CHANGE HISTORY**

77207 First released in Issue 4. Derived from the MSE working draft.

77208 **Issue 6**

77209 The *wcsncat*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

77210 **Issue 8**

77211 Austin Group Defect 448 is applied, adding a requirement that *wcsncat*( ) does not change the
77212 setting of *errno* on valid input.

77213 Austin Group Defect 986 is applied, adding *wcslcat*( ) to the SEE ALSO section.

77214 **NAME**

77215 wcsncmp — compare part of two wide-character strings

77216 **SYNOPSIS**

77217 ```
#include <wchar.h>
```

77218 ```
int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
```

77219 **DESCRIPTION**

77220 CX The functionality described on this reference page is aligned with the ISO C standard. Any
77221 conflict between the requirements described here and the ISO C standard is unintentional. This
77222 volume of POSIX.1-2024 defers to the ISO C standard.

77223 The *wcsncmp*( ) function shall compare not more than *n* wide-character codes (wide-character
77224 codes that follow a null wide-character code are not compared) from the array pointed to by *ws1*
77225 to the array pointed to by *ws2*.

77226 The sign of a non-zero return value shall be determined by the sign of the difference between the
77227 values of the first pair of wide-character codes that differ in the objects being compared.

77228 CX The *wcsncmp*( ) function shall not change the setting of *errno* on valid input.

77229 **RETURN VALUE**

77230 Upon successful completion, *wcsncmp*( ) shall return an integer greater than, equal to, or less
77231 than 0, if the possibly null-terminated array pointed to by *ws1* is greater than, equal to, or less
77232 than the possibly null-terminated array pointed to by *ws2*, respectively.

77233 **ERRORS**

77234 No errors are defined.

77235 **EXAMPLES**

77236 None.

77237 **APPLICATION USAGE**

77238 None.

77239 **RATIONALE**

77240 None.

77241 **FUTURE DIRECTIONS**

77242 None.

77243 **SEE ALSO**

77244 *wcscasecmp*( ), *wcscmp*( )

77245 XBD **<wchar.h>**

77246 **CHANGE HISTORY**

77247 First released in Issue 4. Derived from the MSE working draft.

77248 **Issue 8**

77249 Austin Group Defect 448 is applied, adding a requirement that *wcsncmp*( ) does not change the
77250 setting of *errno* on valid input.

77251 **NAME**

77252 wcpncpy, wcsncpy — copy a fixed-size wide-character string, returning a pointer to its end

77253 **SYNOPSIS**

77254
```
#include <wchar.h>
```

77255 CX
```
wchar_t *wcpncpy(wchar_t restrict *ws1, const wchar_t *restrict ws2,
    size_t n);
```
77256
77257
```
wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    size_t n);
```
77258

77259 **DESCRIPTION**

77260 CX For *wcsncpy*( ): The functionality described on this reference page is aligned with the ISO C
77261 standard. Any conflict between the requirements described here and the ISO C standard is
77262 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

77263 CX The *wcpncpy*( ) and *wcsncpy*( ) functions shall copy not more than *n* wide-character codes (wide-
77264 character codes that follow a null wide-character code are not copied) from the array pointed to
77265 by *ws2* to the array pointed to by *ws1*. If copying takes place between objects that overlap, the
77266 behavior is undefined.

77267 If the array pointed to by *ws2* is a wide-character string that is shorter than *n* wide-character
77268 codes, null wide-character codes shall be appended to the copy in the array pointed to by *ws1*,
77269 until *n* wide-character codes in all are written.

77270 CX The *wcsncpy*( ) and *wcpncpy*( ) functions shall not change the setting of *errno* on valid input.

77271 **RETURN VALUE**

77272 CX If any null wide-character codes were written into the destination, the *wcpncpy*( ) function shall
77273 return the address of the first such null wide-character code. Otherwise, it shall return *&ws1*[*n*].

77274 The *wcsncpy*( ) function shall return *ws1*.

77275 No return values are reserved to indicate an error.

77276 **ERRORS**

77277 No errors are defined.

77278 **EXAMPLES**

77279 None.

77280 **APPLICATION USAGE**

77281 If there is no null wide-character code in the first *n* wide-character codes of the array pointed to
77282 by *ws2*, the result is not null-terminated.

77283 **RATIONALE**

77284 None.

77285 **FUTURE DIRECTIONS**

77286 None.

77287 **SEE ALSO**

77288 *strncpy*( ), *wcscpy*( ), *wcslcat*( )

77289 XBD **<wchar.h>**

77290 **CHANGE HISTORY**

77291 First released in Issue 4. Derived from the MSE working draft.

77292 **Issue 6**

77293 The *wcsncpy*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

77294 **Issue 7**

77295 The *wcpncpy*( ) function is added from The Open Group Technical Standard, 2006, Extended API
77296 Set Part 1.

77297 **Issue 8**

77298 Austin Group Defect 448 is applied, adding a requirement that *wcsncpy*( ) and *wcpncpy*( ) do not
77299 change the setting of *errno* on valid input.

77300 Austin Group Defect 986 is applied, adding *wcslcat*( ) to the SEE ALSO section.

77301 **NAME**

77302 wcsnlen — get length of a fixed-sized wide-character string

77303 **SYNOPSIS**

77304 CX
```
#include <wchar.h>
```
77305
```
size_t wcsnlen(const wchar_t *ws, size_t maxlen);
```

77306 **DESCRIPTION**

77307 Refer to *wcslen( )*.

77308 **NAME**

77309 wcsnrtombs — convert wide-character string to multi-byte string

77310 **SYNOPSIS**

77311 CX
```
#include <wchar.h>

size_t wcsnrtombs(char *restrict dst, const wchar_t **restrict src,
    size_t nwc, size_t len, mbstate_t *restrict ps);
```

77314 **DESCRIPTION**

77315 Refer to *wcsrtombs( )*.

77316 **NAME**

77317 wcspbrk — scan a wide-character string for a wide-character code

77318 **SYNOPSIS**

77319
```
#include <wchar.h>
```

77320
```
wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);
```

77321 **DESCRIPTION**

77322 CX The functionality described on this reference page is aligned with the ISO C standard. Any
77323 conflict between the requirements described here and the ISO C standard is unintentional. This
77324 volume of POSIX.1-2024 defers to the ISO C standard.

77325 The *wcspbrk*( ) function shall locate the first occurrence in the wide-character string pointed to by
77326 *ws1* of any wide-character code from the wide-character string pointed to by *ws2*.

77327 CX The *wcspbrk*( ) function shall not change the setting of *errno* on valid input.

77328 **RETURN VALUE**

77329 Upon successful completion, *wcspbrk*( ) shall return a pointer to the wide-character code or a null
77330 pointer if no wide-character code from *ws2* occurs in *ws1*.

77331 **ERRORS**

77332 No errors are defined.

77333 **EXAMPLES**

77334 None.

77335 **APPLICATION USAGE**

77336 None.

77337 **RATIONALE**

77338 None.

77339 **FUTURE DIRECTIONS**

77340 None.

77341 **SEE ALSO**

77342 *wcschr*( ), *wcsrchr*( )

77343 XBD **<wchar.h>**

77344 **CHANGE HISTORY**

77345 First released in Issue 4. Derived from the MSE working draft.

77346 **Issue 8**

77347 Austin Group Defect 448 is applied, adding a requirement that *wcspbrk*( ) does not change the
77348 setting of *errno* on valid input.

77349 **NAME**

77350 wcsrchr — wide-character string scanning operation

77351 **SYNOPSIS**

77352
```
#include <wchar.h>
```

77353
```
wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);
```

77354 **DESCRIPTION**

77355 CX The functionality described on this reference page is aligned with the ISO C standard. Any
77356 conflict between the requirements described here and the ISO C standard is unintentional. This
77357 volume of POSIX.1-2024 defers to the ISO C standard.

77358 The *wcsrchr*( ) function shall locate the last occurrence of *wc* in the wide-character string pointed
77359 to by *ws*. The application shall ensure that the value of *wc* is a character representable as a type
77360 **wchar_t** and a wide-character code corresponding to a valid character in the current locale. The
77361 terminating null wide-character code shall be considered to be part of the wide-character string.

77362 CX The *wcsrchr*( ) function shall not change the setting of *errno* on valid input.

77363 **RETURN VALUE**

77364 Upon successful completion, *wcsrchr*( ) shall return a pointer to the wide-character code or a null
77365 pointer if *wc* does not occur in the wide-character string.

77366 **ERRORS**

77367 No errors are defined.

77368 **EXAMPLES**

77369 None.

77370 **APPLICATION USAGE**

77371 None.

77372 **RATIONALE**

77373 None.

77374 **FUTURE DIRECTIONS**

77375 None.

77376 **SEE ALSO**

77377 *wcschr*( )

77378 XBD **<wchar.h>**

77379 **CHANGE HISTORY**

77380 First released in Issue 4. Derived from the MSE working draft.

77381 **Issue 6**

77382 The normative text is updated to avoid use of the term ``must'' for application requirements.

77383 **Issue 8**

77384 Austin Group Defect 448 is applied, adding a requirement that *wcsrchr*( ) does not change the
77385 setting of *errno* on valid input.

NAME

wcsnrtombs, wcsrtombs — convert a wide-character string to a character string (restartable)

SYNOPSIS

```
#include <wchar.h>

size_t wcsnrtombs(char *restrict dst, const wchar_t **restrict src,
    size_t nwc, size_t len, mbstate_t *restrict ps);
size_t wcsrtombs(char *restrict dst, const wchar_t **restrict src,
    size_t len, mbstate_t *restrict ps);
```

DESCRIPTION

CX For *wcsrtombs*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *wcsrtombs*() function shall convert a sequence of wide characters from the array indirectly pointed to by *src* into a sequence of corresponding characters, beginning in the conversion state described by the object pointed to by *ps*. If *dst* is not a null pointer, the converted characters shall then be stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null wide character, which shall also be stored. Conversion shall stop earlier in the following cases:

- When a code is reached that does not correspond to a valid character
- When the next character would exceed the limit of *len* total bytes to be stored in the array pointed to by *dst* (and *dst* is not a null pointer)

Each conversion shall take place as if by a call to the *wcrtomb*() function.

If *dst* is not a null pointer, the pointer object pointed to by *src* shall be assigned either a null pointer (if conversion stopped due to reaching a terminating null wide character) or the address just past the last wide character converted (if any). If conversion stopped due to reaching a terminating null wide character, the resulting state described shall be the initial conversion state.

If *ps* is a null pointer, the *wcsrtombs*() function shall use its own internal **mbstate_t** object, which is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence.

If called with a null *ps* argument, the *wcsrtombs*() function need not be thread-safe; however, such calls shall avoid data races with calls to *wcsrtombs*() with a non-null argument and with calls to all other functions.

CX The *wcsnrtombs*() function shall be equivalent to the *wcsrtombs*() function, except that the conversion is limited to the first *nwc* wide characters.

If called with a null *ps* argument, the *wcsnrtombs*() function need not be thread-safe; however, such calls shall avoid data races with calls to *wcsnrtombs*() with a non-null argument and with calls to all other functions.

These functions shall not change the setting of *errno* if successful.

The behavior of these functions shall be affected by the *LC_CTYPE* category of the current locale.

The implementation shall behave as if no function defined in System Interfaces volume of POSIX.1-2024 calls these functions.

77428 **RETURN VALUE**

77429 If conversion stops because a code is reached that does not correspond to a valid character, an
77430 encoding error occurs. In this case, these functions shall store the value of the macro [EILSEQ] in
77431 *errno* and return (**size_t**)–1; the conversion state is undefined. Otherwise, these functions shall
77432 return the number of bytes in the resulting character sequence, not including the terminating
77433 null (if any).

77434 **ERRORS**

77435 These functions shall fail if:

77436 [EILSEQ] A wide-character code does not correspond to a valid character.

77437 These functions may fail if:

77438 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

77439 **EXAMPLES**

77440 None.

77441 **APPLICATION USAGE**

77442 None.

77443 **RATIONALE**

77444 None.

77445 **FUTURE DIRECTIONS**

77446 None.

77447 **SEE ALSO**

77448 *mbsinit*( ), *wcrtomb*( )

77449 XBD **<wchar.h>**

77450 **CHANGE HISTORY**

77451 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
77452 (E).

77453 **Issue 6**

77454 In the DESCRIPTION, a note on using this function in a threaded application is added.

77455 Extensions beyond the ISO C standard are marked.

77456 The normative text is updated to avoid use of the term ``must'' for application requirements.

77457 The *wcsrtombs*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

77458 **Issue 7**

77459 Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *wcsrtombs*( ) function
77460 need not be thread-safe if called with a NULL *ps* argument.

77461 Austin Group Interpretation 1003.1-2001 #170 is applied.

77462 The *wcsnrtombs*( ) function is added from The Open Group Technical Standard, 2006, Extended
77463 API Set Part 1.

77464 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0722 [109,105] is applied.

77465 **Issue 8**

77466 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018
77467 standard.

NAME

wcsspn — get the length of a wide substring

SYNOPSIS

```
#include <wchar.h>

size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *wcsspn*( ) function shall compute the length (in wide characters) of the maximum initial segment of the wide-character string pointed to by *ws1* which consists entirely of wide-character codes from the wide-character string pointed to by *ws2*.

CX The *wcsspn*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

The *wcsspn*( ) function shall return the length of the initial substring of *ws1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscspn( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

The RETURN VALUE section is updated to indicate that *wcsspn*( ) returns the length of *ws1* rather that *ws1* itself.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *wcsspn*( ) does not change the setting of *errno* on valid input.

77505 **NAME**

77506 wcsstr — find a wide-character substring

77507 **SYNOPSIS**

77508
```
#include <wchar.h>
```

77509
```
wchar_t *wcsstr(const wchar_t *restrict ws1,
```

77510
```
    const wchar_t *restrict ws2);
```

77511 **DESCRIPTION**

77512 CX The functionality described on this reference page is aligned with the ISO C standard. Any
77513 conflict between the requirements described here and the ISO C standard is unintentional. This
77514 volume of POSIX.1-2024 defers to the ISO C standard.

77515 The *wcsstr*( ) function shall locate the first occurrence in the wide-character string pointed to by
77516 *ws1* of the sequence of wide characters (excluding the terminating null wide character) in the
77517 wide-character string pointed to by *ws2*.

77518 CX The *wcsstr*( ) function shall not change the setting of *errno* on valid input.

77519 **RETURN VALUE**

77520 Upon successful completion, *wcsstr*( ) shall return a pointer to the located wide-character string,
77521 or a null pointer if the wide-character string is not found.

77522 If *ws2* points to a wide-character string with zero length, the function shall return *ws1*.

77523 **ERRORS**

77524 No errors are defined.

77525 **EXAMPLES**

77526 None.

77527 **APPLICATION USAGE**

77528 None.

77529 **RATIONALE**

77530 None.

77531 **FUTURE DIRECTIONS**

77532 None.

77533 **SEE ALSO**

77534 *wcschr*( )

77535 XBD **<wchar.h>**

77536 **CHANGE HISTORY**

77537 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
77538 (E).

77539 **Issue 6**

77540 The *wcsstr*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

77541 **Issue 8**

77542 Austin Group Defect 448 is applied, adding a requirement that *wcsstr*( ) does not change the
77543 setting of *errno* on valid input.

77544 **NAME**

77545 wcstod, wcstof, wcstold — convert a wide-character string to a double-precision number

77546 **SYNOPSIS**

77547
```
#include <wchar.h>

double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr);
float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr);
long double wcstold(const wchar_t *restrict nptr,
    wchar_t **restrict endptr);
```

77552 **DESCRIPTION**

77553 CX The functionality described on this reference page is aligned with the ISO C standard. Any
77554 conflict between the requirements described here and the ISO C standard is unintentional. This
77555 volume of POSIX.1-2024 defers to the ISO C standard.

77556 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to
77557 **double**, **float**, and **long double** representation, respectively. First, they shall decompose the
77558 input wide-character string into three parts:

77559 1. An initial, possibly empty, sequence of white-space wide characters

77560 2. A subject sequence interpreted as a floating-point constant or representing infinity or
77561 NaN

77562 3. A final wide-character string of one or more unrecognized wide-character codes,
77563 including the terminating null wide-character code of the input wide-character string

77564 Then they shall attempt to convert the subject sequence to a floating-point number, and return
77565 the result.

77566 The expected form of the subject sequence is an optional `'+'` or `'−'` sign, then one of the
77567 following:

77568 • A non-empty sequence of decimal digits optionally containing a radix character; then an
77569 optional exponent part consisting of the wide character `'e'` or the wide character `'E'`,
77570 optionally followed by a `'+'` or `'−'` wide character, and then followed by one or more
77571 decimal digits

77572 • A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix
77573 character; then an optional binary exponent part consisting of the wide character `'p'` or
77574 the wide character `'P'`, optionally followed by a `'+'` or `'−'` wide character, and then
77575 followed by one or more decimal digits

77576 • One of INF or INFINITY, or any other wide string equivalent except for case

77577 • One of NAN or NAN(*n-wchar-sequence*$_{opt}$), or any other wide string ignoring case in the
77578 NAN part, where:

77579
```
n-wchar-sequence:
    digit
    nondigit
    n-wchar-sequence digit
    n-wchar-sequence nondigit
```

77584 The subject sequence is defined as the longest initial subsequence of the input wide string,
77585 starting with the first non-white-space wide character, that is of the expected form. The subject
77586 sequence contains no wide characters if the input wide string is not of the expected form.

77587 If the subject sequence has the expected form for a floating-point number, the sequence of wide
77588 characters starting with the first digit or the radix character (whichever occurs first) shall be

77589 interpreted as a floating constant according to the rules of the C language, except that the radix
77590 character shall be used in place of a period, and that if neither an exponent part nor a radix
77591 character appears in a decimal floating-point number, or if a binary exponent part does not
77592 appear in a hexadecimal floating-point number, an exponent part of the appropriate type with
77593 value zero shall be assumed to follow the last digit in the string. If the subject sequence begins
77594 with a <hyphen-minus>, the sequence shall be interpreted as negated. A wide-character
77595 sequence INF or INFINITY shall be interpreted as an infinity, if representable in the return type,
77596 else as if it were a floating constant that is too large for the range of the return type. A wide-
77597 character sequence NAN or NAN(*n-wchar-sequence*$_{opt}$) shall be interpreted as a quiet NaN, if
77598 supported in the return type, else as if it were a subject sequence part that does not have the
77599 expected form; the meaning of the *n*-wchar sequences is implementation-defined. A pointer to
77600 the final wide string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not
77601 a null pointer.

77602 If the subject sequence has the hexadecimal form and FLT_RADIX is a power of 2, the
77603 conversion shall be rounded in an implementation-defined manner.

77604 CX The radix character shall be as defined in the current locale (category *LC_NUMERIC*). In the
77605 POSIX locale, or in a locale where the radix character is not defined, the radix character shall
77606 default to a <period> (`'.'`).

77607 CX In other than the C or POSIX locale, additional locale-specific subject sequence forms may be
77608 accepted.

77609 If the subject sequence is empty or does not have the expected form, no conversion shall be
77610 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that
77611 *endptr* is not a null pointer.

77612 These functions shall not change the setting of *errno* if successful.

77613 Since 0 is returned on error and is also a valid return on success, an application wishing to check
77614 for error situations should set *errno* to 0, then call *wcstod*( ), *wcstof*( ), or *wcstold*( ), then check
77615 *errno*.

77616 **RETURN VALUE**

77617 Upon successful completion, these functions shall return the converted value. If no conversion
77618 CX could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

77619 If the correct value would cause an overflow and default rounding is in effect, ±HUGE_VAL,
77620 ±HUGE_VALF, or ±HUGE_VALL shall be returned (according to the sign of the value), and *errno*
77621 shall be set to [ERANGE].

77622 If the correct value would cause underflow, a value whose magnitude is no greater than the
77623 CX smallest normalized positive number in the return type shall be returned and *errno* set to
77624 [ERANGE].

77625 **ERRORS**

77626 The *wcstod*( ) function shall fail if:

77627 [ERANGE] The value to be returned would cause overflow and default rounding is in
77628 CX effect or the value to be returned would cause underflow.

77629 The *wcstod*( ) function may fail if:

77630 CX [EINVAL] No conversion could be performed.

EXAMPLES

None.

APPLICATION USAGE

If the subject sequence has the hexadecimal form and FLT_RADIX is not a power of 2, and the result is not exactly representable, the result should be one of the two numbers in the appropriate internal format that are adjacent to the hexadecimal floating source value, with the extra stipulation that the error should have a correct sign for the current rounding direction.

If the subject sequence has the decimal form and at most DECIMAL_DIG (defined in **<float.h>**) significant digits, the result should be correctly rounded. If the subject sequence *D* has the decimal form and more than DECIMAL_DIG significant digits, consider the two bounding, adjacent decimal strings *L* and *U*, both having DECIMAL_DIG significant digits, such that the values of *L*, *D*, and *U* satisfy `"L <= D <= U"`. The result should be one of the (equal or adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current rounding direction, with the extra stipulation that the error with respect to *D* should have a correct sign for the current rounding direction.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fscanf( ), *iswspace*( ), *localeconv*( ), *setlocale*( ), *wcstol*( )

XBD Chapter 7 (on page 127), **<float.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *wcstod*( ) prototype is updated.
- The *wcstof*( ) and *wcstold*( ) functions are added.
- If the correct value for *wcstod*( ) would cause underflow, the return value changed from 0 (as specified in Issue 5) to the smallest normalized positive number.
- The DESCRIPTION, RETURN VALUE, and APPLICATION USAGE sections are extensively updated.

ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/66 is applied, correcting the second paragraph in the RETURN VALUE section.

77673 **Issue 7**

77674 Austin Group Interpretation 1003.1-2001 #015 is applied.

77675 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0723 [302] and XSH/TC1-2008/0724
77676 [105] are applied.

77677 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0391 [584] and XSH/TC2-2008/0392
77678 [796] are applied.

77679 **Issue 8**

77680 Austin Group Defect 1163 is applied, clarifying the handling of white space in the input string.

77681 Austin Group Defect 1686 is applied, addressing some inconsistencies with *strtod*( ).

NAME

wcstoimax, wcstoumax — convert a wide-character string to an integer type

SYNOPSIS

```
#include <stddef.h>
#include <inttypes.h>

intmax_t wcstoimax(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
uintmax_t wcstoumax(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall be equivalent to the *wcstol*( ), *wcstoll*( ), *wcstoul*( ), and *wcstoull*( ) functions, respectively, except that the initial portion of the wide string shall be converted to **intmax_t** and **uintmax_t** representation, respectively.

RETURN VALUE

These functions shall return the converted value, if any.

If no conversion could be performed, zero shall be returned. If the correct value is outside the range of representable values, {INTMAX_MAX}, {INTMAX_MIN}, or {UINTMAX_MAX} shall be returned (according to the return type and sign of the value, if any), and *errno* shall be set to [ERANGE].

ERRORS

These functions shall fail if:

[EINVAL] The value of *base* is not supported.

[ERANGE] The value to be returned is not representable.

These functions may fail if:

[EINVAL] No conversion could be performed.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcstol( ), *wcstoul*( )

XBD **<inttypes.h>**, **<stddef.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

77723 **NAME**

77724 wcstok — split a wide-character string into tokens

77725 **SYNOPSIS**

77726
```
#include <wchar.h>
```

77727
```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict ws2,
```
77728
```
    wchar_t **restrict ptr);
```

77729 **DESCRIPTION**

77730 CX The functionality described on this reference page is aligned with the ISO C standard. Any
77731 conflict between the requirements described here and the ISO C standard is unintentional. This
77732 volume of POSIX.1-2024 defers to the ISO C standard.

77733 A sequence of calls to *wcstok*( ) shall break the wide-character string pointed to by *ws1* into a
77734 sequence of tokens, each of which shall be delimited by a wide-character code from the wide-
77735 character string pointed to by *ws2*. The *ptr* argument points to a caller-provided **wchar_t** pointer
77736 into which the *wcstok*( ) function shall store information necessary for it to continue scanning the
77737 same wide-character string.

77738 The first call in the sequence has *ws1* as its first argument, and is followed by calls with a null
77739 pointer as their first argument. The separator string pointed to by *ws2* may be different from call
77740 to call.

77741 The first call in the sequence shall search the wide-character string pointed to by *ws1* for the first
77742 wide-character code that is *not* contained in the current separator string pointed to by *ws2*. If no
77743 such wide-character code is found, then there are no tokens in the wide-character string pointed
77744 to by *ws1* and *wcstok*( ) shall return a null pointer. If such a wide-character code is found, it shall
77745 be the start of the first token.

77746 The *wcstok*( ) function shall then search from there for a wide-character code that *is* contained in
77747 the current separator string. If no such wide-character code is found, the current token extends
77748 to the end of the wide-character string pointed to by *ws1*, and subsequent searches for a token
77749 shall return a null pointer. If such a wide-character code is found, it shall be overwritten by a
77750 null wide character, which terminates the current token. The *wcstok*( ) function shall save a
77751 pointer to the following wide-character code, from which the next search for a token shall start.

77752 Each subsequent call, with a null pointer as the value of the first argument, shall start searching
77753 from the saved pointer and behave as described above.

77754 The implementation shall behave as if no function calls *wcstok*( ).

77755 CX The *wcstok*( ) function shall not change the setting of *errno* on valid input.

77756 **RETURN VALUE**

77757 Upon successful completion, the *wcstok*( ) function shall return a pointer to the first wide-
77758 character code of a token. Otherwise, if there is no token, *wcstok*( ) shall return a null pointer.

77759 **ERRORS**

77760 No errors are defined.

77761 **EXAMPLES**

77762 None.

77763 **APPLICATION USAGE**

77764 None.

77765 **RATIONALE**

77766 None.

77767 **FUTURE DIRECTIONS**

77768 None.

77769 **SEE ALSO**

77770 XBD **<wchar.h>**

77771 **CHANGE HISTORY**

77772 First released in Issue 4.

77773 **Issue 5**

77774 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, a third argument is
77775 added to the definition of *wcstok*( ) in the SYNOPSIS.

77776 **Issue 6**

77777 The *wcstok*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

77778 **Issue 8**

77779 Austin Group Defect 448 is applied, adding a requirement that *wcstok*( ) does not change the
77780 setting of *errno* on valid input.

NAME

wcstol, wcstoll — convert a wide-character string to a long integer

SYNOPSIS

```
#include <wchar.h>

long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endptr,
    int base);
long long wcstoll(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to **long** and **long long**, respectively. First, they shall decompose the input string into three parts:

1. An initial, possibly empty, sequence of white-space wide characters
2. A subject sequence interpreted as an integer represented in some radix determined by the value of *base*
3. A final wide-character string of one or more unrecognized wide-character codes, including the terminating null wide-character code of the input wide-character string

Then they shall attempt to convert the subject sequence to an integer, and return the result.

If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '−' sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by *base*, optionally preceded by a '+' or '−' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of *base* shall be permitted. If the value of *base* is 16, the wide-character code representations of 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input wide-character string, starting with the first non-white-space wide character, that is of the expected form. The subject sequence contains no wide-character codes if the input wide-character string is empty or consists entirely of white-space wide characters, or if the first non-white-space wide character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes starting with the first digit shall be interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a <hyphen-minus>, the resulting value shall be the negative of the converted value. A pointer to the final wide-character string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

77826 CX In other than the C or POSIX locale, additional locale-specific subject sequence forms may be
77827 accepted.

77828 If the subject sequence is empty or does not have the expected form, no conversion shall be
77829 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that
77830 *endptr* is not a null pointer.

77831 These functions shall not change the setting of *errno* if successful.

77832 Since 0, {LONG_MIN} or {LLONG_MIN} and {LONG_MAX} or {LLONG_MAX} are returned on
77833 error and are also valid returns on success, an application wishing to check for error situations
77834 should set *errno* to 0, then call *wcstol*( ) or *wcstoll*( ), then check *errno*.

77835 **RETURN VALUE**

77836 Upon successful completion, these functions shall return the converted value, if any. If no
77837 CX conversion could be performed, 0 shall be returned and *errno* may be set to indicate the error. If
77838 the correct value is outside the range of representable values, {LONG_MIN}, {LONG_MAX},
77839 {LLONG_MIN}, or {LLONG_MAX} shall be returned (according to the sign of the value), and
77840 *errno* set to [ERANGE].

77841 **ERRORS**

77842 These functions shall fail if:

77843 CX [EINVAL] The value of *base* is not supported.

77844 [ERANGE] The value to be returned is not representable.

77845 These functions may fail if:

77846 CX [EINVAL] No conversion could be performed.

77847 **EXAMPLES**

77848 None.

77849 **APPLICATION USAGE**

77850 None.

77851 **RATIONALE**

77852 None.

77853 **FUTURE DIRECTIONS**

77854 None.

77855 **SEE ALSO**

77856 *fscanf*( ), *iswalpha*( ), *wcstod*( )

77857 XBD **<wchar.h>**

77858 **CHANGE HISTORY**

77859 First released in Issue 4. Derived from the MSE working draft.

77860 **Issue 5**

77861 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

77862 **Issue 6**

77863 Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *wcstol*( ) prototype is updated.
- The *wcstoll*( ) function is added.

Issue 7

SD5-XSH-ERN-56 is applied, removing the reference to **unsigned long** and **unsigned long long** from the DESCRIPTION.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0725 [105] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0393 [584] and XSH/TC2-2008/0394 [796] are applied.

Issue 8

Austin Group Defect 700 is applied, clarifying how a subject sequence beginning with <hyphen-minus> is converted.

Austin Group Defect 1163 is applied, clarifying the handling of white space in the input string.

77881 **NAME**

77882 wcstold — convert a wide-character string to a double-precision number

77883 **SYNOPSIS**

77884 ```
#include <wchar.h>
```

77885 ```
long double wcstold(const wchar_t *restrict nptr,
```

77886 ```
    wchar_t **restrict endptr);
```

77887 **DESCRIPTION**

77888 Refer to *wcstod( )*.

  
```

77889 **NAME**

77890 wcstoll — convert a wide-character string to a long integer

77891 **SYNOPSIS**

77892 ```
#include <wchar.h>
```

77893 ```
long long wcstoll(const wchar_t *restrict nptr,
```

77894 ```
    wchar_t **restrict endptr, int base);
```

77895 **DESCRIPTION**

77896 Refer to *wcstol( )*.

  
```

77897 **NAME**

77898 wcstombs — convert a wide-character string to a character string

77899 **SYNOPSIS**

77900
```
#include <stdlib.h>
```

77901
```
size_t wcstombs(char *restrict s, const wchar_t *restrict pwcs,
```

77902
```
    size_t n);
```

77903 **DESCRIPTION**

77904 CX The functionality described on this reference page is aligned with the ISO C standard. Any
77905 conflict between the requirements described here and the ISO C standard is unintentional. This
77906 volume of POSIX.1-2024 defers to the ISO C standard.

77907 The *wcstombs*( ) function shall convert the sequence of wide-character codes that are in the array
77908 pointed to by *pwcs* into a sequence of characters that begins in the initial shift state and store
77909 these characters into the array pointed to by *s*, stopping if a character would exceed the limit of *n*
77910 total bytes or if a null byte is stored. Each wide-character code shall be converted as if by a call to
77911 *wctomb*( ), except that the shift state of *wctomb*( ) shall not be affected.

77912 The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

77913 No more than *n* bytes shall be modified in the array pointed to by *s*. If copying takes place
77914 CX between objects that overlap, the behavior is undefined. If *s* is a null pointer, *wcstombs*( ) shall
77915 return the length required to convert the entire array regardless of the value of *n*, but no values
77916 are stored.

77917 **RETURN VALUE**

77918 If a wide-character code is encountered that does not correspond to a valid character (of one or
77919 more bytes each), *wcstombs*( ) shall return (**size_t**)–1. Otherwise, *wcstombs*( ) shall return the
77920 number of bytes stored in the character array, not including any terminating null byte. The array
77921 shall not be null-terminated if the value returned is *n*.

77922 **ERRORS**

77923 The *wcstombs*( ) function shall fail if:

77924 CX [EILSEQ] A wide-character code does not correspond to a valid character.

77925 **EXAMPLES**

77926 None.

77927 **APPLICATION USAGE**

77928 None.

77929 **RATIONALE**

77930 None.

77931 **FUTURE DIRECTIONS**

77932 None.

77933 **SEE ALSO**

77934 *mblen*( ), *mbtowc*( ), *mbstowcs*( ), *wctomb*( )

77935 XBD **<stdlib.h>**

77936 **CHANGE HISTORY**

77937 First released in Issue 4. Derived from the ISO C standard.

77938 **Issue 6**

77939 The following new requirements on POSIX implementations derive from alignment with the
77940 Single UNIX Specification:

77941 • The DESCRIPTION states the effect of when *s* is a null pointer.

77942 • The [EILSEQ] error condition is added.

77943 The *wcstombs*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

77944 **Issue 7**

77945 Austin Group Interpretations 1003.1-2001 #156 and #170 are applied.

77946 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0726 [109] is applied.

77947 **NAME**

77948 wcstoul, wcstoull — convert a wide-character string to an unsigned long

77949 **SYNOPSIS**

77950
```
#include <wchar.h>
```

77951
```
unsigned long wcstoul(const wchar_t *restrict nptr,
77952     wchar_t **restrict endptr, int base);
77953 unsigned long long wcstoull(const wchar_t *restrict nptr,
77954     wchar_t **restrict endptr, int base);
```

77955 **DESCRIPTION**

77956 CX The functionality described on this reference page is aligned with the ISO C standard. Any
77957 conflict between the requirements described here and the ISO C standard is unintentional. This
77958 volume of POSIX.1-2024 defers to the ISO C standard.

77959 The *wcstoul*() and *wcstoull*() functions shall convert the initial portion of the wide-character
77960 string pointed to by *nptr* to **unsigned long** and **unsigned long long** representation, respectively.
77961 First, they shall decompose the input wide-character string into three parts:

77962 1. An initial, possibly empty, sequence of white-space wide characters

77963 2. A subject sequence interpreted as an integer represented in some radix determined by the
77964 value of *base*

77965 3. A final wide-character string of one or more unrecognized wide-character codes,
77966 including the terminating null wide-character code of the input wide-character string

77967 Then they shall attempt to convert the subject sequence to an unsigned integer, and return the
77968 result.

77969 If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,
77970 or hexadecimal constant, any of which may be preceded by a `'+'` or `'−'` sign. A decimal
77971 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal
77972 constant consists of the prefix `'0'` optionally followed by a sequence of the digits `'0'` to `'7'`
77973 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the
77974 decimal digits and letters `'a'` (or `'A'`) to `'f'` (or `'F'`) with values 10 to 15 respectively.

77975 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence
77976 of letters and digits representing an integer with the radix specified by *base*, optionally preceded
77977 by a `'+'` or `'−'` sign, but not including an integer suffix. The letters from `'a'` (or `'A'`) to `'z'`
77978 (or `'Z'`) inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less
77979 than that of *base* shall be permitted. If the value of *base* is 16, the wide-character codes 0x or 0X
77980 may optionally precede the sequence of letters and digits, following the sign if present.

77981 The subject sequence is defined as the longest initial subsequence of the input wide-character
77982 string, starting with the first non-white-space wide character, that is of the expected form. The
77983 subject sequence contains no wide-character codes if the input wide-character string is empty or
77984 consists entirely of white-space wide characters, or if the first non-white-space wide character is
77985 other than a sign or a permissible letter or digit.

77986 If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes
77987 starting with the first digit shall be interpreted as an integer constant. If the subject sequence has
77988 the expected form and the value of *base* is between 2 and 36, it shall be used as the base for
77989 conversion, ascribing to each letter its value as given above. If the subject sequence begins with a
77990 <hyphen-minus>, the resulting value shall be the negative of the converted value; this action
77991 shall be performed in the return type. A pointer to the final wide-character string shall be stored
77992 in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

77993 CX In other than the C or POSIX locale, additional locale-specific subject sequence forms may be
77994 accepted.

77995 If the subject sequence is empty or does not have the expected form, no conversion shall be
77996 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that
77997 *endptr* is not a null pointer.

77998 These functions shall not change the setting of *errno* if successful.

77999 Since 0, {ULONG_MAX}, and {ULLONG_MAX} are returned on error and 0 is also a valid return
78000 on success, an application wishing to check for error situations should set *errno* to 0, then call
78001 *wcstoul*( ) or *wcstoull*( ), then check *errno*.

78002 **RETURN VALUE**

78003 Upon successful completion, the *wcstoul*( ) and *wcstoull*( ) functions shall return the converted
78004 CX value, if any. If no conversion could be performed, 0 shall be returned and *errno* may be set to
78005 indicate the error. If the correct value is outside the range of representable values,
78006 {ULONG_MAX} or {ULLONG_MAX} respectively shall be returned and *errno* set to [ERANGE].

78007 **ERRORS**

78008 These functions shall fail if:

78009 CX [EINVAL] The value of *base* is not supported.

78010 [ERANGE] The value to be returned is not representable.

78011 These functions may fail if:

78012 CX [EINVAL] No conversion could be performed.

78013 **EXAMPLES**

78014 None.

78015 **APPLICATION USAGE**

78016 None.

78017 **RATIONALE**

78018 None.

78019 **FUTURE DIRECTIONS**

78020 None.

78021 **SEE ALSO**

78022 *fscanf*( ), *iswalpha*( ), *wcstod*( ), *wcstol*( )

78023 XBD **<wchar.h>**

78024 **CHANGE HISTORY**

78025 First released in Issue 4. Derived from the MSE working draft.

78026 **Issue 5**

78027 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

78028 **Issue 6**

78029 Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EINVAL] error condition is added for when the value of *base* is not supported.

  In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The *wcstoul*( ) prototype is updated.
- The *wcstoull*( ) function is added.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0727 [105] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0395 [584] and XSH/TC2-2008/0396 [796] are applied.

Issue 8

Austin Group Defect 700 is applied, clarifying how a subject sequence beginning with <hyphen-minus> is converted.

Austin Group Defect 1163 is applied, clarifying the handling of white space in the input string.

78046 **NAME**

78047 wcstoumax — convert a wide-character string to an integer type

78048 **SYNOPSIS**

78049 ```
#include <stddef.h>
```

78050 ```
#include <inttypes.h>
```

78051 ```
uintmax_t wcstoumax(const wchar_t *restrict nptr,
```

78052 ```
    wchar_t **restrict endptr, int base);
```

78053 **DESCRIPTION**

78054 Refer to *wcstoimax( )*.

78055 **NAME**

78056 wcswidth — number of column positions of a wide-character string

78057 **SYNOPSIS**

78058 XSI
```
#include <wchar.h>

int wcswidth(const wchar_t *pwcs, size_t n);
```

78060 **DESCRIPTION**

78061 The *wcswidth*( ) function shall determine the number of column positions required for *n* wide-
78062 character codes (or fewer than *n* wide-character codes if a null wide-character code is
78063 encountered before *n* wide-character codes are exhausted) in the string pointed to by *pwcs*.

78064 The *wcswidth*( ) function shall not change the setting of *errno* on valid input.

78065 **RETURN VALUE**

78066 The *wcswidth*( ) function either shall return 0 (if *pwcs* points to a null wide-character code), or
78067 return the number of column positions to be occupied by the wide-character string pointed to by
78068 *pwcs*, or return –1 (if any of the first *n* wide-character codes in the wide-character string pointed
78069 to by *pwcs* is not a printable wide-character code).

78070 **ERRORS**

78071 No errors are defined.

78072 **EXAMPLES**

78073 None.

78074 **APPLICATION USAGE**

78075 This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the
78076 return value for a non-printable wide character is not specified.

78077 **RATIONALE**

78078 None.

78079 **FUTURE DIRECTIONS**

78080 None.

78081 **SEE ALSO**

78082 *wcwidth*( )

78083 XBD Section 3.75 (on page 42), **<wchar.h>**

78084 **CHANGE HISTORY**

78085 First released in Issue 4. Derived from the MSE working draft.

78086 **Issue 6**

78087 The Open Group Corrigendum U021/11 is applied. The function is marked as an extension.

78088 **Issue 8**

78089 Austin Group Defect 448 is applied, adding a requirement that *wcswidth*( ) does not change the
78090 setting of *errno* on valid input.

NAME

wcsxfrm, wcsxfrm_l — wide-character string transformation

SYNOPSIS

```
#include <wchar.h>

size_t wcsxfrm(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    size_t n);
CX  size_t wcsxfrm_l(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    size_t n, locale_t locale);
```

DESCRIPTION

CX For *wcsxfrm*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

CX The *wcsxfrm*( ) and *wcsxfrm_l*( ) functions shall transform the wide-character string pointed to by *ws2* and place the resulting wide-character string into the array pointed to by *ws1*. The transformation shall be such that if *wcscmp*( ) is applied to two transformed wide strings, it shall CX return a value greater than, equal to, or less than 0, corresponding to the result of *wcscoll*( ) and *wcscoll_l*( ) applied to the same two original wide-character strings, and the same *LC_COLLATE* CX category of the current locale or the locale object *locale*, respectively. No more than *n* wide-character codes shall be placed into the resulting array pointed to by *ws1*, including the terminating null wide-character code. If *n* is 0, *ws1* is permitted to be a null pointer. If copying takes place between objects that overlap, the behavior is undefined.

CX The *wcsxfrm*( ) and *wcsxfrm_l*( ) functions shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *wcsxfrm*( ) or *wcsxfrm_l*( ), then check *errno*.

The behavior is undefined if the *locale* argument to *wcsxfrm_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

CX The *wcsxfrm*( ) and *wcsxfrm_l*( ) functions shall return the length of the transformed wide-character string (not including the terminating null wide-character code). If the value returned is *n* or more, the contents of the array pointed to by *ws1* are unspecified.

CX On error, the *wcsxfrm*( ) and *wcsxfrm_l*( ) functions may set *errno*, but no return value is reserved to indicate an error.

ERRORS

These functions may fail if:

CX [EINVAL] The wide-character string pointed to by *ws2* contains wide-character codes outside the domain of the collating sequence.

78127 **EXAMPLES**

78128 None.

78129 **APPLICATION USAGE**

78130 The transformation function is such that two transformed wide-character strings can be ordered
78131 by *wcscmp*( ) as appropriate to collating sequence information in the current locale (category
78132 *LC_COLLATE*).

78133 The fact that when *n* is 0 *ws1* is permitted to be a null pointer is useful to determine the size of
78134 the *ws1* array prior to making the transformation.

78135 **RATIONALE**

78136 None.

78137 **FUTURE DIRECTIONS**

78138 None.

78139 **SEE ALSO**

78140 *wcscmp*( ), *wcscoll*( )

78141 XBD **<wchar.h>**

78142 **CHANGE HISTORY**

78143 First released in Issue 4. Derived from the MSE working draft.

78144 **Issue 5**

78145 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

78146 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

78147 **Issue 6**

78148 In earlier versions, this function was required to return –1 on error.

78149 Extensions beyond the ISO C standard are marked.

78150 The following new requirements on POSIX implementations derive from alignment with the
78151 Single UNIX Specification:

78152 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
78153 added if no conversion could be performed.

78154 The *wcsxfrm*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

78155 **Issue 7**

78156 The *wcsxfrm_l*( ) function is added from The Open Group Technical Standard, 2006, Extended
78157 API Set Part 4.

78158 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0728 [302], XSH/TC1-2008/0729 [283],
78159 XSH/TC1-2008/0730 [283], and XSH/TC1-2008/0731 [302] are applied.

NAME

wctob — wide-character to single-byte conversion

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int wctob(wint_t c);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *wctob*( ) function shall determine whether *c* corresponds to a member of the extended character set whose character representation is a single byte when in the initial shift state.

The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

CX The *wctob*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

The *wctob*( ) function shall return EOF if *c* does not correspond to a character with length one in the initial shift state. Otherwise, it shall return the single-byte representation of that character as an **unsigned char** converted to **int**.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

btowc( )

XBD **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *wctob*( ) does not change the setting of *errno* on valid input.

78197 **NAME**

78198 wctomb — convert a wide-character code to a character

78199 **SYNOPSIS**

78200 `#include <stdlib.h>`

78201 `int wctomb(char *s, wchar_t wchar);`

78202 **DESCRIPTION**

78203 CX Except for requirements relating to data races, the functionality described on this reference page
78204 is aligned with the ISO C standard. Any other conflict between the requirements described here
78205 and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C
78206 standard for all *wctomb*( ) functionality except in relation to data races.

78207 The *wctomb*( ) function shall determine the number of bytes needed to represent the character
78208 corresponding to the wide-character code whose value is *wchar* (including any change in the
78209 shift state). It shall store the character representation (possibly multiple bytes and any special
78210 bytes to change shift state) in the array object pointed to by *s* (if *s* is not a null pointer). At most
78211 {MB_CUR_MAX} bytes shall be stored. If *wchar* is 0, a null byte shall be stored, preceded by any
78212 shift sequence needed to restore the initial shift state, and *wctomb*( ) shall be left in the initial shift
78213 state.

78214 The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a
78215 state-dependent encoding, this function shall be placed into its initial state by a call for which its
78216 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null
78217 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a
78218 null pointer shall cause this function to return a non-zero value if encodings have state
78219 dependency, and 0 otherwise. Changing the *LC_CTYPE* category causes the shift state of this
78220 function to be unspecified.

78221 CX The *wctomb*( ) function need not be thread-safe; however, it shall avoid data races with all other
78222 functions.

78223 The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls
78224 *wctomb*( ).

78225 **RETURN VALUE**

78226 If *s* is a null pointer, *wctomb*( ) shall return a non-zero or 0 value, if character encodings,
78227 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *wctomb*( )
78228 shall return –1 if the value of *wchar* does not correspond to a valid character, or return the
78229 number of bytes that constitute the character corresponding to the value of *wchar*.

78230 In no case shall the value returned be greater than the value of the {MB_CUR_MAX} macro.

78231 **ERRORS**

78232 The *wctomb*( ) function shall fail if:

78233 CX [EILSEQ] An invalid wide-character code is detected.

78234 **EXAMPLES**

78235 None.

78236 **APPLICATION USAGE**

78237 None.

78238 **RATIONALE**

78239 When the ISO C standard introduced threads in C11, it required *wctomb*( ) to avoid data races
78240 (with itself as well as with other functions), whereas POSIX.1-2008 did not require it to be
78241 thread-safe, and in many implementations it did not avoid data races with itself and still does
78242 not. The ISO C committee intend to change the requirements in a future version of the ISO C
78243 standard, but since POSIX.1 currently refers to C17 it is necessary for it not to defer to the ISO C
78244 standard regarding data races in order to continue to allow this function not to avoid data races
78245 with itself.

78246 **FUTURE DIRECTIONS**

78247 It is expected that a change in a future version of the ISO C standard will allow a future version
78248 of this standard to remove the data race exception from the statement that it defers to the ISO C
78249 standard.

78250 **SEE ALSO**

78251 *mblen*( ), *mbtowc*( ), *mbstowcs*( ), *wcstombs*( )

78252 XBD **<stdlib.h>**

78253 **CHANGE HISTORY**

78254 First released in Issue 4. Derived from the ANSI C standard.

78255 **Issue 6**

78256 Extensions beyond the ISO C standard are marked.

78257 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

78258 **Issue 7**

78259 Austin Group Interpretations 1003.1-2001 #156 and #170 are applied.

78260 **Issue 8**

78261 Austin Group Defects 708 and 1302 are applied, aligning this function with the
78262 ISO/IEC 9899:2018 standard, except in relation to data races.

78263 Austin Group Defect 1572 is applied, removing CX shading from some text derived from the
78264 ISO C standard.

NAME

wctrans, wctrans_l — define character mapping

SYNOPSIS

```
#include <wctype.h>

wctrans_t wctrans(const char *charclass);
wctrans_t wctrans_l(const char *charclass, locale_t locale);
```

DESCRIPTION

For *wctrans*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *wctrans*() and *wctrans_l*() functions are defined for valid character mapping names identified in the current locale. The *charclass* is a string identifying a generic character mapping name for which codeset-specific information is required. The following character mapping names are defined in all locales: **tolower** and **toupper**.

These functions shall return a value of type **wctrans_t**, which can be used as the second argument to subsequent calls of *towctrans*() and *towctrans_l*().

The *wctrans*() and *wctrans_l*() functions shall determine values of **wctrans_t** according to the rules of the coded character set defined by character mapping information in the current locale or in the locale represented by *locale*, respectively (category *LC_CTYPE*).

The values returned by *wctrans*() shall be valid until a call to *setlocale*() that modifies the category *LC_CTYPE*.

The values returned by *wctrans_l*() shall be valid only in calls to *towctrans_l*() with a locale represented by *locale* with the same *LC_CTYPE* category value.

The behavior is undefined if the *locale* argument to *wctrans_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *wctrans*() and *wctrans_l*() functions shall return 0 and may set *errno* to indicate the error if the given character mapping name is not valid for the current locale (category *LC_CTYPE*); otherwise, they shall return a non-zero object of type **wctrans_t** that can be used in calls to *towctrans*() and *towctrans_l*().

ERRORS

These functions may fail if:

[EINVAL] The character mapping name pointed to by *charclass* is not valid in the current locale.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

78305 **FUTURE DIRECTIONS**

78306 None.

78307 **SEE ALSO**

78308 *towctrans( )*

78309 XBD **<wctype.h>**

78310 **CHANGE HISTORY**

78311 First released in Issue 5. Derived from ISO/IEC 9899: 1990/Amendment 1: 1995 (E).

78312 **Issue 7**

78313 The *wctrans_l( )* function is added from The Open Group Technical Standard, 2006, Extended
78314 API Set Part 4.

78315 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0732 [302], XSH/TC1-2008/0733 [289],
78316 XSH/TC1-2008/0734 [283], and XSH/TC1-2008/0735 [283] are applied.

NAME

wctype, wctype_l — define character class

SYNOPSIS

```
#include <wctype.h>

wctype_t wctype(const char *property);
wctype_t wctype_l(const char *property, locale_t locale);
```

DESCRIPTION

For *wctype*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *wctype*( ) and *wctype_l*( ) functions are defined for valid character class names as defined in the current locale or in the locale represented by *locale*, respectively.

The *property* argument is a string identifying a generic character class for which codeset-specific type information is required. The following character class names shall be defined in all locales:

alnum	**digit**	**punct**
alpha	**graph**	**space**
blank	**lower**	**upper**
cntrl	**print**	**xdigit**

Additional character class names defined in the locale definition file (category *LC_CTYPE*) can also be specified.

These functions shall return a value of type **wctype_t**, which can be used as the second argument to subsequent calls of *iswctype*( ) and *iswctype_l*( ).

The *wctype*( ) and *wctype_l*( ) functions shall determine values of **wctype_t** according to the rules of the coded character set defined by character type information in the current locale or in the locale represented by *locale*, respectively (category *LC_CTYPE*).

The values returned by *wctype*( ) shall be valid until a call to *setlocale*( ) that modifies the category *LC_CTYPE*.

The values returned by *wctype_l*( ) shall be valid only in calls to *iswctype_l*( ) with a locale represented by *locale* with the same *LC_CTYPE* category value.

The behavior is undefined if the *locale* argument to *wctype_l*( ) is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *wctype*( ) and *wctype_l*( ) functions shall return 0 if the given character class name is not valid for the current locale (category *LC_CTYPE*); otherwise, they shall return an object of type **wctype_t** that can be used in calls to *iswctype*( ) and *iswctype_l*( ).

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswctype( )

XBD **<wctype.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 7

The *wctype_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0736 [302], XSH/TC1-2008/0737 [283], and XSH/TC1-2008/0738 [283] are applied.

NAME

wcwidth — number of column positions of a wide-character code

SYNOPSIS

XSI
```
#include <wchar.h>

int wcwidth(wchar_t wc);
```

DESCRIPTION

The *wcwidth*( ) function shall determine the number of column positions required for the wide character *wc*. The application shall ensure that the value of *wc* is a character representable as a **wchar_t**, and is a wide-character code corresponding to a valid character in the current locale.

The *wcwidth*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

The *wcwidth*( ) function shall either return 0 (if *wc* is a null wide-character code), or return the number of column positions to be occupied by the wide-character code *wc*, or return –1 (if *wc* does not correspond to a printable wide-character code).

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the return value for a non-printable wide character is not specified.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcswidth( )

XBD **<wchar.h>**

CHANGE HISTORY

First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working draft.

Issue 6

The Open Group Corrigendum U021/12 is applied. This function is marked as an extension.

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *wcwidth*( ) does not change the setting of *errno* on valid input.

78414 **NAME**

78415 wmemchr — find a wide character in memory

78416 **SYNOPSIS**

78417 ```
#include <wchar.h>
```

78418 ```
wchar_t *wmemchr(const wchar_t *ws, wchar_t wc, size_t n);
```

78419 **DESCRIPTION**

78420 CX The functionality described on this reference page is aligned with the ISO C standard. Any
78421 conflict between the requirements described here and the ISO C standard is unintentional. This
78422 volume of POSIX.1-2024 defers to the ISO C standard.

78423 The *wmemchr*( ) function shall locate the first occurrence of *wc* in the initial *n* wide characters of
78424 the object pointed to by *ws*. This function shall not be affected by locale and all **wchar_t** values
78425 shall be treated identically. The null wide character and **wchar_t** values not corresponding to
78426 valid characters shall not be treated specially.

78427 If *n* is zero, the application shall ensure that *ws* is a valid pointer and the function behaves as if
78428 no valid occurrence of *wc* is found.

78429 CX The *wmemchr*( ) function shall not change the setting of *errno* on valid input.

78430 **RETURN VALUE**

78431 The *wmemchr*( ) function shall return a pointer to the located wide character, or a null pointer if
78432 the wide character does not occur in the object.

78433 **ERRORS**

78434 No errors are defined.

78435 **EXAMPLES**

78436 None.

78437 **APPLICATION USAGE**

78438 None.

78439 **RATIONALE**

78440 None.

78441 **FUTURE DIRECTIONS**

78442 None.

78443 **SEE ALSO**

78444 *wmemcmp*( ), *wmemcpy*( ), *wmemmove*( ), *wmemset*( )

78445 XBD **<wchar.h>**

78446 **CHANGE HISTORY**

78447 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
78448 (E).

78449 **Issue 6**

78450 The normative text is updated to avoid use of the term ``must'' for application requirements.

78451 **Issue 8**

78452 Austin Group Defect 448 is applied, adding a requirement that *wmemchr*( ) does not change the
78453 setting of *errno* on valid input.

NAME

wmemcmp — compare wide characters in memory

SYNOPSIS

```
#include <wchar.h>

int wmemcmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *wmemcmp*( ) function shall compare the first *n* wide characters of the object pointed to by *ws1* to the first *n* wide characters of the object pointed to by *ws2*. This function shall not be affected by locale and all **wchar_t** values shall be treated identically. The null wide character and **wchar_t** values not corresponding to valid characters shall not be treated specially.

If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function shall behave as if the two objects compare equal.

CX The *wmemcmp*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

The *wmemcmp*( ) function shall return an integer greater than, equal to, or less than zero, respectively, as the object pointed to by *ws1* is greater than, equal to, or less than the object pointed to by *ws2*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wmemchr( ), *wmemcpy*( ), *wmemmove*( ), *wmemset*( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *wmemcmp*( ) does not change the setting of *errno* on valid input.

NAME

wmemcpy — copy wide characters in memory

SYNOPSIS

```
#include <wchar.h>

wchar_t *wmemcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *wmemcpy*( ) function shall copy *n* wide characters from the object pointed to by *ws2* to the object pointed to by *ws1*. This function shall not be affected by locale and all **wchar_t** values shall be treated identically. The null wide character and **wchar_t** values not corresponding to valid characters shall not be treated specially.

If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function shall copy zero wide characters.

CX The *wmemcpy*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

The *wmemcpy*( ) function shall return the value of *ws1*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wmemchr( ), *wmemcmp*( ), *wmemmove*( ), *wmemset*( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

The *wmemcpy*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *wmemcpy*( ) does not change the setting of *errno* on valid input.

NAME

wmemmove — copy wide characters in memory with overlapping areas

SYNOPSIS

```
#include <wchar.h>

wchar_t *wmemmove(wchar_t *ws1, const wchar_t *ws2, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *wmemmove*( ) function shall copy *n* wide characters from the object pointed to by *ws2* to the object pointed to by *ws1*. Copying shall take place as if the *n* wide characters from the object pointed to by *ws2* are first copied into a temporary array of *n* wide characters that does not overlap the objects pointed to by *ws1* or *ws2*, and then the *n* wide characters from the temporary array are copied into the object pointed to by *ws1*.

This function shall not be affected by locale and all **wchar_t** values shall be treated identically. The null wide character and **wchar_t** values not corresponding to valid characters shall not be treated specially.

If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function shall copy zero wide characters.

CX The *wmemmove*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

The *wmemmove*( ) function shall return the value of *ws1*.

ERRORS

No errors are defined

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wmemchr( ), *wmemcmp*( ), *wmemcpy*( ), *wmemset*( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

78576 **Issue 8**

78577 Austin Group Defect 448 is applied, adding a requirement that *wmemmove*( ) does not change the
78578 setting of *errno* on valid input.

NAME

wmemset — set wide characters in memory

SYNOPSIS

```
#include <wchar.h>

wchar_t *wmemset(wchar_t *ws, wchar_t wc, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

The *wmemset*( ) function shall copy the value of *wc* into each of the first *n* wide characters of the object pointed to by *ws*. This function shall not be affected by locale and all **wchar_t** values shall be treated identically. The null wide character and **wchar_t** values not corresponding to valid characters shall not be treated specially.

If *n* is zero, the application shall ensure that *ws* is a valid pointer, and the function shall copy zero wide characters.

CX The *wmemset*( ) function shall not change the setting of *errno* on valid input.

RETURN VALUE

The *wmemset*( ) functions shall return the value of *ws*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wmemchr( ), *wmemcmp*( ), *wmemcpy*( ), *wmemmove*( )

XBD **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

Issue 8

Austin Group Defect 448 is applied, adding a requirement that *wmemset*( ) does not change the setting of *errno* on valid input.

NAME

wordexp, wordfree — perform word expansions

SYNOPSIS

```
#include <wordexp.h>

int wordexp(const char *restrict words, wordexp_t *restrict pwordexp,
    int flags);
void wordfree(wordexp_t *pwordexp);
```

DESCRIPTION

The *wordexp*( ) function shall perform word expansions as described in XCU Section 2.6 (on page 2483), subject to quoting as described in XCU Section 2.2 (on page 2472), and place the list of expanded words into the structure pointed to by *pwordexp*.

The *words* argument is a pointer to a string containing one or more words to be expanded. The expansions shall be the same as would be performed by the command line interpreter if *words* were the part of a command line representing the arguments to a utility. Therefore, the application shall ensure that *words* does not contain an unquoted <newline> character or any of the unquoted shell special characters `'|'`, `'&'`, `';'`, `'<'`, `'>'` except in the context of command substitution as specified in XCU Section 2.6.3 (on page 2489). It also shall not contain unquoted parentheses or braces, except in the context of command or variable substitution. The application shall ensure that every member of *words* which it expects to have expanded by *wordexp*( ) does not contain an unquoted initial comment character. The application shall also ensure that any words which it intends to be ignored (because they begin or continue a comment) are deleted from *words*. If the argument *words* contains an unquoted comment character (<number-sign>) that is the beginning of a token, *wordexp*( ) shall either treat the comment character as a regular character, or interpret it as a comment indicator and ignore the remainder of *words*.

The structure type **wordexp_t** is defined in the **<wordexp.h>** header and includes at least the following members:

Member Type	Member Name	Description
size_t	*we_wordc*	Count of words matched by *words*.
**char ** **	*we_wordv*	Pointer to list of expanded words.
size_t	*we_offs*	Slots to reserve at the beginning of *pwordexp*−>*we_wordv*.

The *wordexp*( ) function shall store the number of generated words into *pwordexp*−>*we_wordc* and a pointer to a list of pointers to words in *pwordexp*−>*we_wordv*. Each individual field created during field splitting (see XCU Section 2.6.5, on page 2491) or pathname expansion (see XCU Section 2.6.6, on page 2493) shall be a separate word in the *pwordexp*−>*we_wordv* list. The words shall be in order as described in XCU Section 2.6 (on page 2483). The first pointer after the last word pointer shall be a null pointer. The expansion of special parameters described in XCU Section 2.5.2 (on page 2479) is unspecified.

It is the caller's responsibility to allocate the storage pointed to by *pwordexp*. The *wordexp*( ) function shall allocate other space as needed, including memory pointed to by *pwordexp*−>*we_wordv*. The *wordfree*( ) function shall free any memory associated with *pwordexp* from a previous call to *wordexp*( ). The *wordfree*( ) function shall not modify *errno* if *pwordexp* was previously modified by *wordexp*( ) and not yet freed.

The *flags* argument is used to control the behavior of *wordexp*( ). The value of *flags* is the bitwise-inclusive OR of zero or more of the following constants, which are defined in **<wordexp.h>**:

78664 WRDE_APPEND Append words generated to the ones from a previous call to *wordexp*( ).

78665 WRDE_DOOFFS Make use of *pwordexp*−>*we_offs*. If this flag is set, *pwordexp*−>*we_offs* is
78666 used to specify how many null pointers to add to the beginning of
78667 *pwordexp*−>*we_wordv*. In other words, *pwordexp*−>*we_wordv* shall point to
78668 *pwordexp*−>*we_offs* null pointers, followed by *pwordexp*−>*we_wordc* word
78669 pointers, followed by a null pointer.

78670 WRDE_NOCMD If the implementation supports the utilities defined in the Shell and
78671 Utilities volume of POSIX.1-2024, fail if command substitution, as
78672 specified in XCU Section 2.6.3 (on page 2489), is requested.

78673 WRDE_REUSE The *pwordexp* argument was passed to a previous successful call to
78674 *wordexp*( ), and has not been passed to *wordfree*( ). The result shall be the
78675 same as if the application had called *wordfree*( ) and then called *wordexp*( )
78676 without WRDE_REUSE.

78677 WRDE_SHOWERR Do not redirect *stderr* to **/dev/null**.

78678 WRDE_UNDEF Report error on an attempt to expand an undefined shell variable.

78679 The WRDE_APPEND flag can be used to append a new set of words to those generated by a
78680 previous call to *wordexp*( ). The following rules apply to applications when two or more calls to
78681 *wordexp*( ) are made with the same value of *pwordexp* and without intervening calls to *wordfree*( ):

78682 1. The first such call shall not set WRDE_APPEND. All subsequent calls shall set it.

78683 2. All of the calls shall set WRDE_DOOFFS, or all shall not set it.

78684 3. After the second and each subsequent call, *pwordexp*−>*we_wordv* shall point to a list
78685 containing the following:

78686 a. Zero or more null pointers, as specified by WRDE_DOOFFS and
78687 *pwordexp*−>*we_offs*

78688 b. Pointers to the words that were in the *pwordexp*−>*we_wordv* list before the call, in
78689 the same order as before

78690 c. Pointers to the new words generated by the latest call, in the specified order

78691 4. The count returned in *pwordexp*−>*we_wordc* shall be the total number of words from all of
78692 the calls.

78693 5. The application can change any of the fields after a call to *wordexp*( ), but if it does it shall
78694 reset them to the original value before a subsequent call, using the same *pwordexp* value,
78695 to *wordfree*( ) or *wordexp*( ) with the WRDE_APPEND or WRDE_REUSE flag.

78696 If the implementation supports the utilities defined in the Shell and Utilities volume of
78697 POSIX.1-2024, and *words* contains an unquoted character—<newline>, `'|'`, `'&'`, `';'`, `'<'`, `'>'`,
78698 `'('`, `')'`, `'{'`, `'}'`—in an inappropriate context, *wordexp*( ) shall fail, and the number of
78699 expanded words shall be 0.

78700 Unless WRDE_SHOWERR is set in *flags*, *wordexp*( ) shall redirect *stderr* to **/dev/null** for any
78701 utilities executed as a result of command substitution while expanding *words*. If
78702 WRDE_SHOWERR is set, *wordexp*( ) may write messages to *stderr* if syntax errors are detected
78703 while expanding *words*, unless the *stderr* stream has wide orientation in which case the behavior
78704 is undefined. It is unspecified whether any write errors encountered while outputting such
78705 messages will affect the *stderr* error indicator or the value of *errno*.

78706 The application shall ensure that if WRDE_DOOFFS is set, then *pwordexp*−>*we_offs* has the same

78707 value for each *wordexp*( ) call and *wordfree*( ) call using a given *pwordexp*.

78708 The results are unspecified if WRDE_APPEND and WRDE_REUSE are both specified.

78709 The following constants are defined as error return values:

78710 WRDE_BADCHAR One of the unquoted characters—<newline>, '|', '&', ';', '<', '>',
78711 '(', ')', '{', '}'—appears in *words* in an inappropriate context.

78712 WRDE_BADVAL Reference to undefined shell variable when WRDE_UNDEF is set in *flags*.

78713 WRDE_CMDSUB Command substitution requested when WRDE_NOCMD was set in *flags*.

78714 WRDE_NOSPACE Attempt to allocate memory failed.

78715 WRDE_SYNTAX Shell syntax error, such as unbalanced parentheses or unterminated
78716 string.

78717 **RETURN VALUE**

78718 Upon successful completion, *wordexp*( ) shall return 0. Otherwise, a non-zero value, as described
78719 in **<wordexp.h>**, shall be returned to indicate an error. If *wordexp*( ) returns the value
78720 WRDE_NOSPACE, then *pwordexp*->*we_wordc* and *pwordexp*->*we_wordv* shall be updated to
78721 reflect any words that were successfully expanded. In other error cases, if the WRDE_APPEND
78722 flag was specified, *pwordexp*->*we_wordc* and *pwordexp*->*we_wordv* shall not be modified.

78723 The *wordfree*( ) function shall not return a value.

78724 **ERRORS**

78725 No errors are defined.

78726 **EXAMPLES**

78727 None.

78728 **APPLICATION USAGE**

78729 The *wordexp*( ) function is intended to be used by an application that wants to do all of the shell's
78730 expansions on a word or words obtained from a user. For example, if the application prompts
78731 for a pathname (or list of pathnames) and then uses *wordexp*( ) to process the input, the user
78732 could respond with anything that would be valid as input to the shell.

78733 The WRDE_NOCMD flag is provided for applications that, for security or other reasons, want to
78734 prevent a user from executing shell commands. Disallowing unquoted shell special characters
78735 also prevents unwanted side-effects, such as executing a command or writing a file.

78736 POSIX.1-2024 does not require the *wordexp*( ) function to be thread-safe if passed an expression
78737 referencing an environment variable while any other thread is concurrently modifying any
78738 environment variable; see *exec* (on page 866).

78739 Even though the WRDE_SHOWERR flag allows the implementation to write messages to *stderr*
78740 during command substitution or syntax errors, this standard does not provide any way to detect
78741 write failures during the output of such messages.

78742 Applications which use wide-character output functions with *stderr* should ensure that any calls
78743 to *wordexp*( ) do not write to *stderr*, by avoiding use of the WRDE_SHOWERR flag.

78744 **RATIONALE**

78745 This function was included as an alternative to *glob*( ). There had been continuing controversy
78746 over exactly what features should be included in *glob*( ). It is hoped that by providing *wordexp*( )
78747 (which provides all of the shell word expansions, but which may be slow to execute) and *glob*( )
78748 (which is faster, but which only performs pathname expansion, without tilde or parameter
78749 expansion) this will satisfy the majority of applications.

While *wordexp*( ) could be implemented entirely as a library routine, it is expected that most implementations run a shell in a subprocess to do the expansion.

Two different approaches have been proposed for how the required information might be presented to the shell and the results returned. They are presented here as examples.

One proposal is to extend the *echo* utility by adding a **–q** option. This option would cause *echo* to add a <backslash> before each <backslash> and <blank> that occurs within an argument. The *wordexp*( ) function could then invoke the shell as follows:

```
(void) strcpy(buffer, "echo –q");
(void) strcat(buffer, words);
if ((flags & WRDE_SHOWERR) == 0)
    (void) strcat(buffer, "2>/dev/null");
f = popen(buffer, "r");
```

The *wordexp*( ) function would read the resulting output, remove unquoted <backslash> characters, and break into words at unquoted <blank> characters. If the WRDE_NOCMD flag was set, *wordexp*( ) would have to scan *words* before starting the subshell to make sure that there would be no command substitution. In any case, it would have to scan *words* for unquoted special characters.

Another proposal is to add the following options to *sh*:

–w *wordlist*
: This option provides a wordlist expansion service to applications. The words in *wordlist* shall be expanded and the following written to standard output:

  1. The count of the number of words after expansion, in decimal, followed by a null byte
  2. The number of bytes needed to represent the expanded words (not including null separators), in decimal, followed by a null byte
  3. The expanded words, each terminated by a null byte

  If an error is encountered during word expansion, *sh* exits with a non-zero status after writing the former to report any words successfully expanded

–P Run in ``protected'' mode. If specified with the **–w** option, no command substitution shall be performed.

With these options, *wordexp*( ) could be implemented fairly simply by creating a subprocess using *fork*( ) and executing *sh* using the line:

```
execl(<shell path>, "sh", "–P", "–w", words, (char *)0);
```

after directing standard error to **/dev/null**.

It seemed objectionable for a library routine to write messages to standard error, unless explicitly requested, so *wordexp*( ) is required to redirect standard error to **/dev/null** to ensure that no messages are generated, even for commands executed for command substitution. The WRDE_SHOWERR flag can be specified to request that error messages be written.

The WRDE_REUSE flag allows the implementation to avoid the expense of freeing and reallocating memory, if that is possible. A minimal implementation can call *wordfree*( ) when WRDE_REUSE is set.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fnmatch*( ), *glob*( )

XBD **<wordexp.h>**

XCU Chapter 2 (on page 2472)

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 5

Moved from POSIX2 C-language Binding to BASE.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

The **restrict** keyword is added to the *wordexp*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #148 is applied, adding APPLICATION USAGE.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0739 [460], XSH/TC1-2008/0740 [291], and XSH/TC1-2008/0741 [460] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0397 [608], XSH/TC2-2008/0398 [704], XSH/TC2-2008/0399 [704], and XSH/TC2-2008/0400 [608] are applied.

Issue 8

Austin Group Defect 385 is applied, adding a requirement that *wordfree*( ) does not modify *errno* when passed a pointer to a **wordexp_t** that can be freed.

78814 **NAME**

78815 wprintf — print formatted wide-character output

78816 **SYNOPSIS**

78817 `#include <stdio.h>`

78818 `#include <wchar.h>`

78819 `int wprintf(const wchar_t *restrict format, ...);`

78820 **DESCRIPTION**

78821 Refer to *fwprintf( )*.

78822 **NAME**

78823 pwrite, write — write on a file

78824 **SYNOPSIS**

78825 ```
#include <unistd.h>
```

78826 ```
ssize_t pwrite(int fildes, const void *buf, size_t nbyte,
```

78827 ```
    off_t offset);
```

78828 ```
ssize_t write(int fildes, const void *buf, size_t nbyte);
```

78829 **DESCRIPTION**

78830 The *write*( ) function shall attempt to write *nbyte* bytes from the buffer pointed to by *buf* to the
78831 file associated with the open file descriptor, *fildes*.

78832 Before any action described below is taken, and if *nbyte* is zero and the file is a regular file, the
78833 *write*( ) function may detect and return errors as described below. In the absence of errors, or if
78834 error detection is not performed, the *write*( ) function shall return zero and have no other results.
78835 If *nbyte* is zero and the file is not a regular file, the results are unspecified.

78836 On a regular file or other file capable of seeking, the actual writing of data shall proceed from
78837 the position in the file indicated by the file offset associated with *fildes*. Before successful return
78838 from *write*( ), the file offset shall be incremented by the number of bytes actually written. On a
78839 regular file, if the position of the last byte written is greater than or equal to the length of the file,
78840 the length of the file shall be set to this position plus one.

78841 On a file not capable of seeking, writing shall always take place starting at the current position.
78842 The value of a file offset associated with such a device is undefined.

78843 If the O_APPEND flag of the file status flags is set, the file offset shall be set to the end of the file
78844 prior to each write and no intervening file modification operation shall occur between changing
78845 the file offset and the write operation.

78846 If a *write*( ) requests that more bytes be written than there is room for (for example, the file size
78847 limit of the process or the physical end of a medium), only as many bytes as there is room for
78848 shall be written. For example, suppose there is space for 20 bytes more in a file before reaching a
78849 limit. A write of 512 bytes will return 20. The next write of a non-zero number of bytes would
78850 give a failure return (except as noted below).

78851 If the request would cause the file size to exceed the soft file size limit for the process and there
78852 XSI is no room for any bytes to be written, the request shall fail and the implementation shall
78853 generate a SIGXFSZ signal for the thread.

78854 If *write*( ) is interrupted by a signal before it writes any data, it shall return –1 with *errno* set to
78855 [EINTR].

78856 If *write*( ) is interrupted by a signal after it successfully writes some data, it shall return the
78857 number of bytes written.

78858 If the value of *nbyte* is greater than {SSIZE_MAX}, the result is implementation-defined.

78859 After a *write*( ) to a regular file has successfully returned:

78860 • Any successful *read*( ) from each byte position in the file that was modified by that write
78861 shall return the data specified by the *write*( ) for that position until such byte positions are
78862 again modified.

78863 • Any subsequent successful *write*( ) to the same byte position in the file shall overwrite that
78864 file data.

Write requests to a pipe or FIFO shall be handled in the same way as a regular file with the following exceptions:

- There is no file offset associated with a pipe or FIFO, hence each write request shall append to the end of the pipe or FIFO.
- Write requests of {PIPE_BUF} bytes or less shall not be interleaved with data from other threads performing write operations on the same pipe or FIFO. Writes of greater than {PIPE_BUF} bytes may have data interleaved, on arbitrary boundaries, with write operations by other threads, whether or not the O_NONBLOCK flag of the file status flags is set.
- If the O_NONBLOCK flag is clear, a write request may cause the thread to block, but on normal completion it shall return *nbyte*.
- If the O_NONBLOCK flag is set, *write*( ) requests shall be handled differently, in the following ways:
  - The *write*( ) function shall not block the thread.
  - A write request for {PIPE_BUF} or fewer bytes shall have the following effect: if there is sufficient space available in the pipe or FIFO, *write*( ) shall transfer all the data and return the number of bytes requested. Otherwise, *write*( ) shall transfer no data and return –1 with *errno* set to [EAGAIN].
  - A write request for more than {PIPE_BUF} bytes shall cause one of the following:
    - When at least one byte can be written, transfer what it can and return the number of bytes written. When all data previously written to the pipe or FIFO is read, it shall transfer at least {PIPE_BUF} bytes.
    - When no data can be written, transfer no data, and return –1 with *errno* set to [EAGAIN].

When attempting to write to a file descriptor (other than a pipe or FIFO) that supports non-blocking writes and cannot accept the data immediately:

- If the O_NONBLOCK flag is clear, *write*( ) shall block the calling thread until the data can be accepted.
- If the O_NONBLOCK flag is set, *write*( ) shall not block the thread. If some data can be written without blocking the thread, *write*( ) shall write what it can and return the number of bytes written. Otherwise, it shall return –1 and set *errno* to [EAGAIN].

Upon successful completion, where *nbyte* is greater than 0, *write*( ) shall mark for update the last data modification and last file status change timestamps of the file, and if the file is a regular file, the S_ISUID and S_ISGID bits of the file mode may be cleared.

For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *fildes*.

If *fildes* refers to a socket, *write*( ) shall be equivalent to *send*( ) with no flags set.

SIO If the O_DSYNC bit has been set, write I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion.

If the O_SYNC bit has been set, write I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.

SHM If *fildes* refers to a shared memory object, the result of the *write*( ) function is unspecified.

78907 TYM If *fildes* refers to a typed memory object, the result of the *write*( ) function is unspecified.

78908 The *pwrite*( ) function shall be equivalent to *write*( ), except that it writes into a given position
78909 and does not change the file offset (regardless of whether O_APPEND is set). The first three
78910 arguments to *pwrite*( ) are the same as *write*( ) with the addition of a fourth argument *offset* for
78911 the desired position inside the file. An attempt to perform a *pwrite*( ) on a file that is incapable of
78912 seeking shall result in an error.

78913 **RETURN VALUE**

78914 Upon successful completion, these functions shall return the number of bytes actually written to
78915 the file associated with *fildes*. This number shall never be greater than *nbyte*. Otherwise, –1 shall
78916 be returned and *errno* set to indicate the error.

78917 **ERRORS**

78918 These functions shall fail if:

78919 [EAGAIN] The file is neither a pipe, nor a FIFO, nor a socket, the O_NONBLOCK flag is
78920 set for the file descriptor, and the thread would be delayed in the *write*( )
78921 operation.

78922 [EBADF] The *fildes* argument is not a valid file descriptor open for writing.

78923 [EFBIG] An attempt was made to write a file that exceeds the implementation-defined
78924 maximum file size and there was no room for any bytes to be written.

78925 [EFBIG] An attempt was made to write a file that exceeds the file size limit of the
78926 XSI process, and there was no room for any bytes to be written. A SIGXFSZ
78927 signal shall also be generated for the thread.

78928 [EFBIG] The file is a regular file, *nbyte* is greater than 0, and the starting position is
78929 greater than or equal to the offset maximum established in the open file
78930 description associated with *fildes*.

78931 [EINTR] The write operation was terminated due to the receipt of a signal, and no data
78932 was transferred.

78933 [EIO] The process is a member of a background process group attempting to write to
78934 its controlling terminal, TOSTOP is set, the calling thread is not blocking
78935 SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the
78936 process is orphaned. This error may also be returned under implementation-
78937 defined conditions.

78938 [ENOSPC] There was no free space remaining on the device containing the file.

78939 The *pwrite*( ) function shall fail if:

78940 [EINVAL] The file is a regular file or block special file, and the *offset* argument is
78941 negative. The file offset shall remain unchanged.

78942 [ESPIPE] The file is incapable of seeking.

78943 The *write*( ) function shall fail if:

78944 [EAGAIN] The file is a pipe or FIFO, the O_NONBLOCK flag is set for the file descriptor,
78945 and the thread would be delayed in the write operation.

78946 [EAGAIN] or [EWOULDBLOCK]
78947 The file is a socket, the O_NONBLOCK flag is set for the file descriptor, and
78948 the thread would be delayed in the write operation.

78949 [ECONNRESET] A write was attempted on a socket that is not connected.

78950 [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
78951 any process, or that only has one end open. A SIGPIPE signal shall also be sent
78952 to the thread.

78953 [EPIPE] A write was attempted on a socket that is shut down for writing, or is no
78954 longer connected. In the latter case, if the socket is of type SOCK_STREAM, a
78955 SIGPIPE signal shall also be sent to the thread.

78956 These functions may fail if:

78957 [EIO] A physical I/O error has occurred.

78958 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

78959 [ENXIO] A request was made of a nonexistent device, or the request was outside the
78960 capabilities of the device.

78961 The *write*( ) function may fail if:

78962 [EACCES] A write was attempted on a socket and the calling process does not have
78963 appropriate privileges.

78964 [ENETDOWN] A write was attempted on a socket and the local network interface used to
78965 reach the destination is down.

78966 [ENETUNREACH]
78967 A write was attempted on a socket and no route to the network is present.

78968 **EXAMPLES**

78969 **Writing from a Buffer**

78970 The following example writes data from the buffer pointed to by *buf* to the file associated with
78971 the file descriptor *fd*.

```
78972 #include <sys/types.h>
78973 #include <string.h>
78974 ...
78975 char buf[20];
78976 size_t nbytes;
78977 ssize_t bytes_written;
78978 int fd;
78979 ...
78980 strcpy(buf, "This is a test\n");
78981 nbytes = strlen(buf);
78982 bytes_written = write(fd, buf, nbytes);
78983 ...
```

78984 **APPLICATION USAGE**

78985 None.

78986 **RATIONALE**

78987 See also the RATIONALE section in *read*( ).

78988 An attempt to write to a pipe or FIFO has several major characteristics:

- *Atomic/non-atomic*: A write is atomic if the whole amount written in one operation is not interleaved with data from any other thread. This is useful when there are multiple writers sending data to a single reader. Applications need to know how large a write request can be expected to be performed atomically. This maximum is called {PIPE_BUF}. This volume of POSIX.1-2024 does not say whether write requests for more than {PIPE_BUF} bytes are atomic, but requires that writes of {PIPE_BUF} or fewer bytes shall be atomic.
- *Blocking/immediate*: Blocking is only possible with O_NONBLOCK clear. If there is enough space for all the data requested to be written immediately, the implementation should do so. Otherwise, the calling thread may block; that is, pause until enough space is available for writing. The effective size of a pipe or FIFO (the maximum amount that can be written in one operation without blocking) may vary dynamically, depending on the implementation, so it is not possible to specify a fixed value for it.
- *Complete/partial/deferred*: A write request:

```
int fildes;
size_t nbyte;
ssize_t ret;
char *buf;

ret = write(fildes, buf, nbyte);
```

may return:

Complete *ret*=*nbyte*

Partial *ret*<*nbyte*

This shall never happen if *nbyte*≤{PIPE_BUF}. If it does happen (with *nbyte*>{PIPE_BUF}), this volume of POSIX.1-2024 does not guarantee atomicity, even if *ret*≤{PIPE_BUF}, because atomicity is guaranteed according to the amount *requested*, not the amount *written*.

Deferred: *ret*=–1, *errno*=[EAGAIN]

This error indicates that a later request may succeed. It does not indicate that it *shall* succeed, even if *nbyte*≤{PIPE_BUF}, because if no process reads from the pipe or FIFO, the write never succeeds. An application could usefully count the number of times [EAGAIN] is caused by a particular value of *nbyte*>{PIPE_BUF} and perhaps do later writes with a smaller value, on the assumption that the effective size of the pipe or FIFO may have decreased.

Partial and deferred writes are only possible with O_NONBLOCK set.

The relations of these properties are shown in the following tables:

Write to a Pipe or FIFO with O_NONBLOCK *clear*			
Immediately Writable:	**None**	**Some**	*nbyte*
nbyte≤{PIPE_BUF}	Atomic blocking *nbyte*	Atomic blocking *nbyte*	Atomic immediate *nbyte*
nbyte>{PIPE_BUF}	Blocking *nbyte*	Blocking *nbyte*	Blocking *nbyte*

If the O_NONBLOCK flag is clear, a write request shall block if the amount writable immediately is less than that requested. If the flag is set (by *fcntl*( )), a write request shall never block.

Write to a Pipe or FIFO with O_NONBLOCK *set*			
Immediately Writable:	**None**	**Some**	*nbyte*
nbyte≤{PIPE_BUF}	–1, [EAGAIN]	–1, [EAGAIN]	Atomic *nbyte*
nbyte>{PIPE_BUF}	–1, [EAGAIN]	<*nbyte* or –1, [EAGAIN]	≤*nbyte* or –1, [EAGAIN]

There is no exception regarding partial writes when O_NONBLOCK is set. With the exception of writing to an empty pipe or FIFO, this volume of POSIX.1-2024 does not specify exactly when a partial write is performed since that would require specifying internal details of the implementation. Every application should be prepared to handle partial writes when O_NONBLOCK is set and the requested amount is greater than {PIPE_BUF}, just as every application should be prepared to handle partial writes on other kinds of file descriptors.

The intent of forcing writing at least one byte if any can be written is to assure that each write makes progress if there is any room in the pipe or FIFO. If the pipe or FIFO is empty, {PIPE_BUF} bytes must be written; if not, at least some progress must have been made.

Where this volume of POSIX.1-2024 requires –1 to be returned and *errno* set to [EAGAIN], most historical implementations return zero (with the O_NDELAY flag set, which is the historical predecessor of O_NONBLOCK, but is not itself in this volume of POSIX.1-2024). The error indications in this volume of POSIX.1-2024 were chosen so that an application can distinguish these cases from end-of-file. While *write*( ) cannot receive an indication of end-of-file, *read*( ) can, and the two functions have similar return values. Also, some existing systems (for example, Eighth Edition) permit a write of zero bytes to mean that the reader should get an end-of-file indication; for those systems, a return value of zero from *write*( ) indicates a successful write of an end-of-file indication.

Implementations are allowed, but not required, to perform error checking for *write*( ) requests of zero bytes.

The concept of a {PIPE_MAX} limit (indicating the maximum number of bytes that can be written to a pipe or FIFO in a single operation) was considered, but rejected, because this concept would unnecessarily limit application writing.

See also the discussion of O_NONBLOCK in *read*( ).

Writes can be serialized with respect to other reads and writes. If a *read*( ) of file data can be proven (by any means) to occur after a *write*( ) of the data, it must reflect that *write*( ), even if the calls are made by different threads. A similar requirement applies to multiple write operations to the same file position. This is needed to guarantee the propagation of data from *write*( ) calls to subsequent *read*( ) calls. This requirement is particularly significant for networked file systems, where some caching schemes violate these semantics.

Note that this is specified in terms of *read*( ) and *write*( ). The XSI extensions *readv*( ) and *writev*( ) also obey these semantics. A new ``high-performance'' write analog that did not follow these serialization requirements would also be permitted by this wording. This volume of POSIX.1-2024 is also silent about any effects of application-level caching (such as that done by *stdio*).

This volume of POSIX.1-2024 does not specify the value of the file offset after an error is returned; there are too many cases. For programming errors, such as [EBADF], the concept is meaningless since no file is involved. For errors that are detected immediately, such as [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however, an updated value would be very useful and is the behavior of many implementations.

This volume of POSIX.1-2024 does not specify the behavior of concurrent writes to a regular file

from multiple threads, except that each write is atomic (see Section 2.9.7, on page 547). Applications should use some form of concurrency control.

This volume of POSIX.1-2024 intentionally does not specify any *pwrite*( ) errors related to pipes, FIFOs, and sockets other than [ESPIPE].

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *creat*( ), *dup*( ), *fcntl*( ), *getrlimit*( ), *lseek*( ), *open*( ), *pipe*( ), *read*( ), *writev*( )

XBD **<limits.h>**, **<sys/uio.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Large File Summit extensions are added.

The *pwrite*( ) function is added.

Issue 6

The DESCRIPTION states that the *write*( ) function does not block the thread. Previously this said ``process'' rather than ``thread''.

The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are marked as part of the XSI STREAMS Option Group.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION now states that if *write*( ) is interrupted by a signal after it has successfully written some data, it returns the number of bytes written. In the POSIX.1-1988 standard, it was optional whether *write*( ) returned the number of bytes written, or whether it returned –1 with *errno* set to [EINTR]. This is a FIPS requirement.
- The following changes are made to support large files:
  - For regular files, no data transfer occurs past the offset maximum established in the open file description associated with the *fildes*.
  - A second [EFBIG] error condition is added.
- The [EIO] error condition is added.
- The [EPIPE] error condition is added for when a pipe has only one end open.
- The [ENXIO] optional error condition is added.

Text referring to sockets is added to the DESCRIPTION.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effect of reading zero bytes is clarified.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *write*( ) results are unspecified for typed memory objects.

The following error conditions are added for operations on sockets: [EAGAIN],

[EWOULDBLOCK], [ECONNRESET], [ENOTCONN], and [EPIPE].

The [EIO] error is made optional.

The [ENOBUFS] error is added for sockets.

The following error conditions are added for operations on sockets: [EACCES], [ENETDOWN], and [ENETUNREACH].

The *writev*( ) function is split out into a separate reference page.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/146 is applied, updating text in the ERRORS section from ``a SIGPIPE signal is generated to the calling process'' to ``a SIGPIPE signal shall also be sent to the thread''.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/147 is applied, making a correction to the RATIONALE.

Issue 7

The *pwrite*( ) function is moved from the XSI option to the Base.

Functionality relating to the XSI STREAMS option is marked obsolescent.

SD5-XSH-ERN-160 is applied, updating the DESCRIPTION to clarify the requirements for the *pwrite*( ) function, and to change the use of the phrase ``file pointer'' to ``file offset''.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0742 [219], XSH/TC1-2008/0743 [215], XSH/TC1-2008/0744 [79], and XSH/TC1-2008/0745 [215] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0401 [676,710] and XSH/TC2-2008/0402 [966] are applied.

Issue 8

Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1430 is applied, clarifying that requirements relating to data interleaving on pipes and FIFOs apply to write operations in other threads, not just other processes, and changing some uses of ``pipe'' to ``pipe or FIFO''.

Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error relating to the file size limit for the process.

NAME

writev — write a vector

SYNOPSIS

XSI
```
#include <sys/uio.h>

ssize_t writev(int fildes, const struct iovec *iov, int iovcnt);
```

DESCRIPTION

The *writev*( ) function shall be equivalent to *write*( ), except as described below. The *writev*( ) function shall gather output data from the *iovcnt* buffers specified by the members of the *iov* array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*–1]. The *iovcnt* argument is valid if greater than 0 and less than or equal to {IOV_MAX}, as defined in **<limits.h>**.

Each *iovec* entry specifies the base address and length of an area in memory from which data should be written. The *writev*( ) function shall always write a complete area before proceeding to the next.

If *fildes* refers to a regular file and all of the *iov_len* members in the array pointed to by *iov* are 0, *writev*( ) shall return 0 and have no other effect. For other file types, the behavior is unspecified.

If the sum of the *iov_len* values is greater than {SSIZE_MAX}, the operation shall fail and no data shall be transferred.

RETURN VALUE

Upon successful completion, *writev*( ) shall return the number of bytes actually written. Otherwise, it shall return a value of –1, the file-pointer shall remain unchanged, and *errno* shall be set to indicate an error.

ERRORS

Refer to *write*( ).

In addition, the *writev*( ) function shall fail if:

[EINVAL] The sum of the *iov_len* values in the *iov* array would overflow an **ssize_t**.

The *writev*( ) function may fail and set *errno* to:

[EINVAL] The *iovcnt* argument was less than or equal to 0, or greater than {IOV_MAX}.

EXAMPLES

Writing Data from an Array

The following example writes data from the buffers specified by members of the *iov* array to the file associated with the file descriptor *fd*.

```
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
...
ssize_t bytes_written;
int fd;
char *buf0 = "short string\n";
char *buf1 = "This is a longer string\n";
char *buf2 = "This is the longest string in this example\n";
int iovcnt;
struct iovec iov[3];
```

```
        iov[0].iov_base = buf0;
        iov[0].iov_len = strlen(buf0);
        iov[1].iov_base = buf1;
        iov[1].iov_len = strlen(buf1);
        iov[2].iov_base = buf2;
        iov[2].iov_len = strlen(buf2);
        ...
        iovcnt = sizeof(iov) / sizeof(struct iovec);
        bytes_written = writev(fd, iov, iovcnt);
        ...
```

APPLICATION USAGE

None.

RATIONALE

Refer to *write*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

readv( ), *write*( )

XBD **<limits.h>**, **<sys/uio.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 6

Split out from the *write*( ) reference page.

79211 **NAME**

79212 wscanf — convert formatted wide-character input

79213 **SYNOPSIS**

79214 ```
#include <stdio.h>
```

79215 ```
#include <wchar.h>
```

79216 ```
int wscanf(const wchar_t *restrict format, ...);
```

79217 **DESCRIPTION**

79218 Refer to *fwscanf*( ).

  
```

NAME

y0, y1, yn — Bessel functions of the second kind

SYNOPSIS

XSI
```
#include <math.h>

double y0(double x);
double y1(double x);
double yn(int n, double x);
```

DESCRIPTION

The *y0*( ), *y1*( ), and *yn*( ) functions shall compute Bessel functions of *x* of the second kind of orders 0, 1, and *n*, respectively. *y0*(*x*) shall be equivalent to *yn*(0, *x*), and *y1*(*x*) shall be equivalent to *yn*(1, *x*).

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the relevant Bessel value of *x* of the second kind.

MXX If *x* is NaN, NaN shall be returned.

MXX If the *x* argument to these functions is negative, either NaN (if supported) or the same return value as when *x* is 0.0 (see below) shall be returned, and a domain error may occur.

If *x* is 0.0, *y0*( ) and *y1*( ) shall return –HUGE_VAL and a pole error may occur. If *x* is 0.0 and *n* is not both negative and odd, *yn*( ) shall return –HUGE_VAL and a pole error may occur. If *x* is 0.0 and *n* is negative and odd, *yn*( ) shall return +HUGE_VAL and a pole error may occur.

MXX If *x* is +Inf, +0 shall be returned.

MXX If the correct result would cause underflow and is not representable, a range error may occur,
MXX and the function shall return 0.0, or (if the IEC 60559 Floating-Point option is not supported) an implementation-defined value no greater in magnitude than DBL_MIN.

MXX If the correct result would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

If the correct result of calling *y1*( ) would cause overflow, –HUGE_VAL shall be returned and a range error may occur. If *n* is not both negative and odd, and the correct result of calling *yn*( ) would cause overflow, –HUGE_VAL shall be returned and a range error may occur. If *n* is negative and odd, and the correct result of calling *yn*( ) would cause overflow, +HUGE_VAL shall be returned and a range error may occur.

ERRORS

These functions may fail if:

Domain Error — The value of *x* is negative.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error — The value of *x* is zero.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

Range Error — The value of *x* is too large in magnitude, or the correct result would cause underflow.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The *y1*( ) and *yn*( ) functions may fail if:

Range Error — The correct result would cause overflow.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( ), *j0*( )

XBD Section 4.23 (on page 109), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The normative text is updated to avoid use of the term ``must'' for application requirements.

The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/148 is applied, updating the RETURN VALUE and ERRORS sections. The changes are made for consistency with the general rules stated in ``Treatment of Error Conditions for Mathematical Functions'' in the Base Definitions volume of POSIX.1-2024.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0746 [68] is applied.

Issue 8

Austin Group Defect 714 is applied, changing the behavior of these functions for special cases to be a better match for their mathematical behavior.

79308 *The Open Group Standard*

79309 **Vol. 3:**

79310 **Shell and Utilities, Issue 8**

79311 *The Open Group*
79312 *The Institute of Electrical and Electronics Engineers, Inc.*

Chapter 1 Introduction

The Shell and Utilities volume of POSIX.1-2024 describes the commands and utilities offered to application programs by POSIX-conformant systems.

1.1 Relationship to Other Documents

1.1.1 System Interfaces

This subsection describes some of the features provided by the System Interfaces volume of POSIX.1-2024 that are assumed to be globally available on all systems conforming to this volume of POSIX.1-2024. This subsection does not attempt to detail all of the features defined in the System Interfaces volume of POSIX.1-2024 that are required by all of the utilities defined in this volume of POSIX.1-2024; the utility and function descriptions point out additional functionality required to provide the corresponding specific features needed by each.

The following subsections describe frequently used concepts. Many of these concepts are described in the Base Definitions volume of POSIX.1-2024. Utility and function description statements override these defaults when appropriate.

1.1.1.1 Process Attributes

The following process attributes, as described in the System Interfaces volume of POSIX.1-2024, are assumed to be supported for all processes in this volume of POSIX.1-2024:

Controlling Terminal	Real Group ID
Current Working Directory	Real User ID
Effective Group ID	Root Directory
Effective User ID	Saved Set-Group-ID
File Descriptors	Saved Set-User-ID
File Mode Creation Mask	Session Membership
Process Group ID	Supplementary Group IDs
Process ID	

A conforming implementation may include additional process attributes.

1.1.1.2 Concurrent Execution of Processes

The following functionality of the *fork*( ) function defined in the System Interfaces volume of POSIX.1-2024 shall be available on all systems conforming to this volume of POSIX.1-2024:

1. Independent processes shall be capable of executing independently without either process terminating.

2. A process shall be able to create a new process with all of the attributes referenced in Section 1.1.1.1 (on page 2453), determined according to the semantics of a call to the *fork*( ) function defined in the System Interfaces volume of POSIX.1-2024 followed by a call in the child process to one of the *exec* functions defined in the System Interfaces volume of POSIX.1-2024.

1.1.1.3 *File Access Permissions*

The file access control mechanism described by XBD Section 4.7 (on page 97) shall apply to all files on an implementation conforming to this volume of POSIX.1-2024.

1.1.1.4 *File Read, Write, and Creation*

If a file that does not exist is to be written, it shall be created as described below, unless the utility description states otherwise.

When a file that does not exist is created, the following features defined in the System Interfaces volume of POSIX.1-2024 shall apply unless the utility or function description states otherwise:

1. The user ID of the file shall be set to the effective user ID of the calling process.
2. The group ID of the file shall be set to the effective group ID of the calling process or the group ID of the directory in which the file is being created.
3. If the file is a regular file, the permission bits of the file shall be set to:

   S_IROTH | S_IWOTH | S_IRGRP | S_IWGRP | S_IRUSR | S_IWUSR

   (see the description of *File Modes* in XBD Chapter 14 (on page 221), **<sys/stat.h>**) except that the bits specified by the file mode creation mask of the process shall be cleared. If the file is a directory, the permission bits shall be set to:

   S_IRWXU | S_IRWXG | S_IRWXO

   except that the bits specified by the file mode creation mask of the process shall be cleared.
4. The last data access, last data modification, and last file status change timestamps of the file shall be updated as specified in XBD Section 4.12 (on page 98).
5. If the file is a directory, it shall be an empty directory; otherwise, the file shall have length zero.
6. If the file is a symbolic link, the effect shall be undefined unless the {POSIX2_SYMLINKS} variable is in effect for the directory in which the symbolic link would be created.
7. Unless otherwise specified, the file created shall be a regular file.

When an attempt is made to create a file that already exists, the utility shall take the action indicated in Table 1-1 (on page 2455) corresponding to the type of the file the utility is trying to create and the type of the existing file, unless the utility description states otherwise.

Table 1-1 Actions when Creating a File that Already Exists

Existing Type	New Type B	C	D	F	L	M	P	Q	R	S	T	Function Creating New
B Block Special	F	F	F	F	F	U	U	U	OF	U	U	*mknod*()**
C Character Special	F	F	F	F	F	U	U	U	OF	U	U	*mknod*()**
D Directory	F	F	F	F	F	—	—	—	F	—	U	*mkdir*()
F FIFO Special File	F	F	F	F	F	—	—	—	O	—	U	*mkfifo*()
L Symbolic Link	F	F	F	F	F	—	—	—	FL	—	U	*symlink*()
M Shared Memory	F	F	F	F	F	—	—	—	—	—	U	*shm_open*()
P Semaphore	F	F	F	F	F	—	—	—	—	—	U	*sem_open*()
Q Message Queue	F	F	F	F	F	—	—	—	—	—	U	*mq_open*()
R Regular File	F	F	F	F	F	—	—	—	RF	—	U	*open*()
S Socket	F	F	F	F	F	—	—	—	—	—	U	*bind*()
T Typed Memory	F	F	F	F	F	U	U	U	U	U	U	*

The following codes are used in Table 1-1:

F Fail. The attempt to create the new file shall fail and the utility shall either continue with its operation or exit immediately with an exit status that indicates an error occurred, depending on the description of the utility.

FL Follow link. Unless otherwise specified, the symbolic link shall be followed as specified for pathname resolution, and the operation performed shall be as if the target of the symbolic link (after all resolution) had been named. If the target of the symbolic link does not exist, it shall be as if that nonexistent target had been named directly.

O Open FIFO. When attempting to create a regular file, and the existing file is a FIFO special file:

1. If the FIFO is not already open for reading, the attempt shall block until the FIFO is opened for reading.
2. Once the FIFO is open for reading, the utility shall open the FIFO for writing and continue with its operation.

OF The named file shall be opened with the consequences defined for that file type.

RF Regular file. When attempting to create a regular file, and the existing file is a regular file:

1. The user ID, group ID, and permission bits of the file shall not be changed.
2. The file shall be truncated to zero length.
3. The last data modification and last file status change timestamps shall be marked for update.

— The effect is implementation-defined unless specified by the utility description.

U The effect is unspecified unless specified by the utility description.

* There is no portable way to create a file of this type.

** Not portable.

When a file is to be appended, the file shall be opened in a manner equivalent to using the O_APPEND flag, without the O_TRUNC flag, in the *open*() function defined in the System Interfaces volume of POSIX.1-2024.

When a file is to be read or written, the file shall be opened with an access mode corresponding

to the operation to be performed. If file access permissions deny access, the requested operation shall fail.

1.1.1.5 File Removal

When a directory that is the root directory or current working directory of any process is removed, the effect is implementation-defined. If file access permissions deny access, the requested operation shall fail. Otherwise, when a file is removed:

1. Its directory entry shall be removed from the file system.
2. The link count of the file shall be decremented.
3. If the file is an empty directory (see XBD Section 3.119, on page 48):
    a. If no process has the directory open, the space occupied by the directory shall be freed and the directory shall no longer be accessible.
    b. If one or more processes have the directory open, the directory contents shall be preserved until all references to the file have been closed.
4. If the file is a directory that is not empty, the last file status change timestamp shall be marked for update.
5. If the file is not a directory:
    a. If the link count becomes zero:
        i. If no process has the file open, the space occupied by the file shall be freed and the file shall no longer be accessible.
        ii. If one or more processes have the file open, the file contents shall be preserved until all references to the file have been closed.
    b. If the link count is not reduced to zero, the last file status change timestamp shall be marked for update.
6. The last data modification and last file status change timestamps of the containing directory shall be marked for update.

1.1.1.6 File Time Values

All files shall have the three time values described by XBD Section 4.12 (on page 98).

1.1.1.7 File Contents

When a reference is made to the contents of a file, *pathname*, this means the equivalent of all of the data placed in the space pointed to by *buf* when performing the *read*() function calls in the following operations defined in the System Interfaces volume of POSIX.1-2024:

```
while (read (fildes, buf, nbytes) > 0)
    ;
```

If the file is indicated by a pathname *pathname*, the file descriptor shall be determined by the equivalent of the following operation defined in the System Interfaces volume of POSIX.1-2024:

```
fildes = open (pathname, O_RDONLY);
```

The value of *nbytes* in the above sequence is unspecified; if the file is of a type where the data returned by *read*() would vary with different values, the value shall be one that results in the

most data being returned.

If the *read*( ) function calls would return an error, it is unspecified whether the contents of the file are considered to include any data from offsets in the file beyond where the error would be returned.

1.1.1.8 Pathname Resolution

The pathname resolution algorithm, described by XBD Section 4.16 (on page 105), shall be used by implementations conforming to this volume of POSIX.1-2024; see also XBD Section 4.8 (on page 97).

1.1.1.9 Changing the Current Working Directory

When the current working directory (see XBD Section 3.94, on page 45) is to be changed, unless the utility or function description states otherwise, the operation shall succeed unless a call to the *chdir*( ) function defined in the System Interfaces volume of POSIX.1-2024 would fail when invoked with the new working directory pathname as its argument.

1.1.1.10 Establish the Locale

The functionality of the *setlocale*( ) function defined in the System Interfaces volume of POSIX.1-2024 shall be available on all systems conforming to this volume of POSIX.1-2024; that is, utilities that require the capability of establishing an international operating environment shall be permitted to set the specified category of the international environment.

1.1.1.11 Actions Equivalent to Functions

Some utility descriptions specify that a utility performs actions equivalent to a function defined in the System Interfaces volume of POSIX.1-2024. Such specifications require only that the external effects be equivalent, not that any effect within the utility and visible only to the utility be equivalent.

1.1.2 Concepts Derived from the ISO C Standard

Some of the standard utilities perform complex data manipulation using their own procedure and arithmetic languages, as defined in their EXTENDED DESCRIPTION or OPERANDS sections. Unless otherwise noted, the arithmetic and semantic concepts (precision, type conversion, control flow, and so on) shall be equivalent to those defined in the ISO C standard, as described in the following sections. Note that there is no requirement that the standard utilities be implemented in any particular programming language.

1.1.2.1 Arithmetic Precision and Operations

Integer variables and constants, including the values of operands and option-arguments, used by the standard utilities listed in this volume of POSIX.1-2024 shall be implemented as equivalent to the ISO C standard **signed long** data type; floating point shall be implemented as equivalent to the ISO C standard **double** type. Conversions between types shall be as described in the ISO C standard. All variables shall be initialized to zero if they are not otherwise assigned by the input to the application.

Arithmetic operators and control flow keywords shall be implemented as equivalent to those in

the cited ISO C standard section, as listed in Table 1-2.

Note: The comma operator (section 6.5.17 of the ISO C standard) is intentionally not included in the table. It need not be supported by implementations.

Table 1-2 Selected ISO C Standard Operators and Control Flow Keywords

Operation	ISO C Standard Equivalent Reference
( )	Section 6.5.1, Primary Expressions
postfix ++ postfix --	Section 6.5.2, Postfix Operators
unary + unary - prefix ++ prefix -- ~ ! *sizeof*( )	Section 6.5.3, Unary Operators
* / %	Section 6.5.5, Multiplicative Operators
+ -	Section 6.5.6, Additive Operators
<< >>	Section 6.5.7, Bitwise Shift Operators
<, <= >, >=	Section 6.5.8, Relational Operators
== !=	Section 6.5.9, Equality Operators
&	Section 6.5.10, Bitwise AND Operator
^	Section 6.5.11, Bitwise Exclusive OR Operator
|	Section 6.5.12, Bitwise Inclusive OR Operator
&&	Section 6.5.13, Logical AND Operator
||	Section 6.5.14, Logical OR Operator
expr?*expr*:*expr*	Section 6.5.15, Conditional Operator
=, *=, /=, %=, +=, -= <<=, >>=, &=, ^=, |=	Section 6.5.16, Assignment Operators
if ( ) **if** ( ) . . . **else** **switch** ( )	Section 6.8.4, Selection Statements
while ( ) **do** . . . **while** ( ) **for** ( )	Section 6.8.5, Iteration Statements
goto **continue** **break** **return**	Section 6.8.6, Jump Statements

The evaluation of arithmetic expressions shall be equivalent to that described in Section 6.5, Expressions, of the ISO C standard.

1.1.2.2 *Mathematical Functions*

Any mathematical functions with the same names as those in the following sections of the ISO C standard:

- Section 7.12, Mathematics, `<math.h>`
- Section 7.22.2, Pseudo-Random Sequence Generation Functions

shall be implemented to return the results equivalent to those returned from a call to the corresponding function described in the ISO C standard.

1.2 Utility Limits

This section lists magnitude limitations imposed by a specific implementation. The braces notation, {LIMIT}, is used in this volume of POSIX.1-2024 to indicate these values, but the braces are not part of the name.

Table 1-3 Utility Limit Minimum Values

Name	Description	Value
{POSIX2_BC_BASE_MAX}	The maximum *obase* value allowed by the *bc* utility.	99
{POSIX2_BC_DIM_MAX}	The maximum number of elements permitted in an array by the *bc* utility.	2 048
{POSIX2_BC_SCALE_MAX}	The maximum *scale* value allowed by the *bc* utility.	99
{POSIX2_BC_STRING_MAX}	The maximum length of a string constant accepted by the *bc* utility.	1 000
{POSIX2_COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order** keyword in the locale definition file; see the **border_start** keyword in XBD Section 7.3.2 (on page 139).	2
{POSIX2_EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the *expr* utility.	32
{POSIX2_LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <newline>.	2 048
{POSIX_RE_DUP_MAX}	Maximum number of repeated occurrences of a BRE or ERE interval expression; see XBD Section 9.3.6 (on page 185) and Section 9.4.6 (on page 189).	255

The values specified in Table 1-3 represent the lowest values conforming implementations shall provide and, consequently, the largest values on which an application can rely without further enquiries, as described below. These values shall be accessible to applications via the *getconf*

utility (see *getconf*, on page 2973).

Implementations may provide more liberal, or less restrictive, values than shown in Table 1-3 (on page 2459). These possibly more liberal values are accessible using the symbols in Table 1-4.

The *sysconf*( ) function defined in the System Interfaces volume of POSIX.1-2024 or the *getconf* utility return the value of each symbol on each specific implementation. The value so retrieved is the largest, or most liberal, value that is available throughout the session lifetime, as determined at session creation. The literal names shown in the table apply only to the *getconf* utility; the high-level language binding describes the exact form of each name to be used by the interfaces in that binding.

All numeric limits defined by the System Interfaces volume of POSIX.1-2024, such as {PATH_MAX}, shall also apply to this volume of POSIX.1-2024. All the utilities defined by this volume of POSIX.1-2024 are implicitly limited by these values, unless otherwise noted in the utility descriptions.

It is not guaranteed that the application can actually reach the specified limit of an implementation in any given case, or at all, as a lack of virtual memory or other resources may prevent this. The limit value indicates only that the implementation does not specifically impose any arbitrary, more restrictive limit.

Table 1-4 Symbolic Utility Limits

Name	Description	Minimum Value
{BC_BASE_MAX}	The maximum *obase* value allowed by the *bc* utility.	{POSIX2_BC_BASE_MAX}
{BC_DIM_MAX}	The maximum number of elements permitted in an array by the *bc* utility.	{POSIX2_BC_DIM_MAX}
{BC_SCALE_MAX}	The maximum *scale* value allowed by the *bc* utility.	{POSIX2_BC_SCALE_MAX}
{BC_STRING_MAX}	The maximum length of a string constant accepted by the *bc* utility.	{POSIX2_BC_STRING_MAX}
{COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order** keyword in the locale definition file; see the **order_start** keyword in XBD Section 7.3.2 (on page 139).	{POSIX2_COLL_WEIGHTS_MAX}
{EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the *expr* utility.	{POSIX2_EXPR_NEST_MAX}

Name	Description	Minimum Value
{LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <newline>.	{POSIX2_LINE_MAX}
{RE_DUP_MAX}	Maximum number of repeated occurrences of a BRE or ERE interval expression; see XBD Section 9.3.6 (on page 185) and Section 9.4.6 (on page 189).	{POSIX_RE_DUP_MAX}

The following value may be a constant within an implementation or may vary from one pathname to another.

{POSIX2_SYMLINKS}
When referring to a directory, the system supports the creation of symbolic links within that directory; for non-directory files, the meaning of {POSIX2_SYMLINKS} is undefined.

1.3 Grammar Conventions

Portions of this volume of POSIX.1-2024 are expressed in terms of a special grammar notation. It is used to portray the complex syntax of certain program input. The grammar is based on the syntax used by the *yacc* utility. However, it does not represent fully functional *yacc* input, suitable for program use; the lexical processing and all semantic requirements are described only in textual form. The grammar is not based on source used in any traditional implementation and has not been tested with the semantic code that would normally be required to accompany it. Furthermore, there is no implication that the partial *yacc* code presented represents the most efficient, or only, means of supporting the complex syntax within the utility. Implementations may use other programming languages or algorithms, as long as the syntax supported is the same as that represented by the grammar.

The following typographical conventions are used in the grammar; they have no significance except to aid in reading.

- The identifiers for the reserved words of the language are shown with a leading capital letter. (These are terminals in the grammar; for example, **While**, **Case**.)
- The identifiers for terminals in the grammar are all named with uppercase letters and underscores; for example, **NEWLINE**, **ASSIGN_OP**, **NAME**.
- The identifiers for non-terminals are all lowercase.

1.4 Utility Description Defaults

This section describes all of the subsections used within the utility descriptions, including:

- Intended usage of the section
- Global defaults that affect all the standard utilities
- The meanings of notations used in this volume of POSIX.1-2024 that are specific to individual utility sections

NAME

This section gives the name or names of the utility and briefly states its purpose.

SYNOPSIS

The SYNOPSIS section summarizes the syntax of the calling sequence for the utility, including options, option-arguments, and operands. Standards for utility naming are described in XBD Section 12.2 (on page 215); for describing the utility's arguments in XBD Section 12.1 (on page 213).

DESCRIPTION

The DESCRIPTION section describes the actions of the utility. If the utility has a very complex set of subcommands or its own procedural language, an EXTENDED DESCRIPTION section is also provided. Most explanations of optional functionality are omitted here, as they are usually explained in the OPTIONS section.

As stated in Section 1.1.1.11 (on page 2457), some functions are described in terms of equivalent functionality. When specific functions are cited, the implementation shall provide equivalent functionality including side-effects associated with successful execution of the function. The treatment of errors and intermediate results from the individual functions cited is generally not specified by this volume of POSIX.1-2024. See the utility's EXIT STATUS and CONSEQUENCES OF ERRORS sections for all actions associated with errors encountered by the utility.

A standard utility shall not be treated as a declaration utility unless explicitly stated in this section.

OPTIONS

The OPTIONS section describes the utility options and option-arguments, and how they modify the actions of the utility. Standard utilities that have options either fully comply with XBD Section 12.2 (on page 215) or describe all deviations. Apparent disagreements between functionality descriptions in the OPTIONS and DESCRIPTION (or EXTENDED DESCRIPTION) sections are always resolved in favor of the OPTIONS section.

Each OPTIONS section that uses the phrase ``The ... utility shall conform to the Utility Syntax Guidelines ...'' refers only to the use of the utility as specified by this volume of POSIX.1-2024; implementation extensions should also conform to the guidelines, but may allow exceptions for historical practice.

Unless otherwise stated in the utility description, when given an option unrecognized by the implementation, or when a required option-argument is not provided, standard utilities shall issue a diagnostic message to standard error and exit with an exit status that indicates an error occurred.

All utilities in this volume of POSIX.1-2024 shall be capable of processing arguments using eight-bit transparency.

Default Behavior: When this section is listed as ``None.'', it means that the implementation need not support any options. Standard utilities that do not accept

79709 options, but that do accept operands, shall recognize `"--"` as a first argument to be
79710 discarded.

79711 The requirement for recognizing `"--"` is because conforming applications need a way
79712 to shield their operands from any arbitrary options that the implementation may
79713 provide as an extension. For example, if the standard utility *foo* is listed as taking no
79714 options, and the application needed to give it a pathname with a leading <hyphen-
79715 minus>, it could safely do it as:

79716 ```
foo -- -myfile
```

79717 and avoid any problems with –**m** used as an extension.

79718 **OPERANDS**

79719 The OPERANDS section describes the utility operands, and how they affect the actions
79720 of the utility. Apparent disagreements between functionality descriptions in the
79721 OPERANDS and DESCRIPTION (or EXTENDED DESCRIPTION) sections shall be
79722 resolved in favor of the OPERANDS section.

79723 If an operand naming a file can be specified as `'-'`, which means to use the standard
79724 input instead of a named file, this is explicitly stated in this section. Unless otherwise
79725 stated, the use of multiple instances of `'-'` to mean standard input in a single
79726 command produces unspecified results.

79727 Unless otherwise stated, the standard utilities that accept operands shall process those
79728 operands in the order specified in the command line.

79729 **Default Behavior:** When this section is listed as ``None.'', it means that the
79730 implementation need not support any operands.

79731 **STDIN**

79732 The STDIN section describes the standard input of the utility. This section is frequently
79733 merely a reference to the following section, as many utilities treat standard input and
79734 input files in the same manner. Unless otherwise stated, all restrictions described in the
79735 INPUT FILES section shall apply to this section as well.

79736 Use of a terminal for standard input can cause any of the standard utilities that read
79737 standard input to stop when used in the background. For this reason, applications
79738 should not use interactive features in scripts to be placed in the background.

79739 The specified standard input format of the standard utilities shall not depend on the
79740 existence or value of the environment variables defined in this volume of POSIX.1-2024,
79741 except as provided by this volume of POSIX.1-2024.

79742 **Default Behavior:** When this section is listed as ``Not used.'', it means that the standard
79743 input shall not be read when the utility is used as described by this volume of
79744 POSIX.1-2024.

79745 **INPUT FILES**

79746 The INPUT FILES section describes the files, other than the standard input, used as
79747 input by the utility. It includes files named as operands and option-arguments as well
79748 as other files that are referred to, such as start-up and initialization files, databases, and
79749 so on. Commonly-used files are generally described in one place and cross-referenced
79750 by other utilities.

79751 All utilities in this volume of POSIX.1-2024 shall be capable of processing input files
79752 using eight-bit transparency.

79753 When a standard utility reads a seekable input file and terminates without an error
79754 before it reaches end-of-file, the utility shall ensure that the file offset in the open file

  
```

description is properly positioned just past the last byte processed by the utility. For files that are not seekable, the state of the file offset in the open file description for that file is unspecified. A conforming application shall not assume that the following three commands are equivalent:

```
tail -n +2 file
(sed -n 1q; cat) < file
cat file | (sed -n 1q; cat)
```

The second command is equivalent to the first only when the file is seekable. The third command leaves the file offset in the open file description in an unspecified state. Other utilities, such as *head*, *read*, and *sh*, have similar properties.

Some of the standard utilities, such as filters, process input files a line or a block at a time and have no restrictions on the maximum input file size. Some utilities may have size limitations that are not as obvious as file space or memory limitations. Such limitations should reflect resource limitations of some sort, not arbitrary limits set by implementors. Implementations shall document those utilities that are limited by constraints other than file system space, available memory, and other limits specifically cited by this volume of POSIX.1-2024, and identify what the constraint is and indicate a way of estimating when the constraint would be reached. Similarly, some utilities descend the directory tree (recursively). Implementations shall also document any limits that they may have in descending the directory tree that are beyond limits cited by this volume of POSIX.1-2024.

When an input file is described as a ``text file'', the utility produces undefined results if given input that is not from a text file, unless otherwise stated. Some utilities (for example, *make*, *read*, *sh*) allow for continued input lines using an escaped <newline> convention; unless otherwise stated, the utility need not be able to accumulate more than {LINE_MAX} bytes from a set of multiple, continued input lines. Thus, for a conforming application the total of all the continued lines in a set cannot exceed {LINE_MAX}. If a utility using the escaped <newline> convention detects an end-of-file condition immediately after an escaped <newline>, the results are unspecified.

Record formats are described in a notation similar to that used by the C-language function, *printf*( ). See XBD Chapter 5 (on page 113) for a description of this notation. The format description is intended to be sufficiently rigorous to allow other applications to generate these input files. However, since <blank>s can legitimately be included in some of the fields described by the standard utilities, particularly in locales other than the POSIX locale, this intent is not always realized.

Default Behavior: When this section is listed as ``None.'', it means that no input files are required to be supplied when the utility is used as described by this volume of POSIX.1-2024.

ENVIRONMENT VARIABLES

The ENVIRONMENT VARIABLES section lists what variables affect the utility's execution.

The entire manner in which environment variables described in this volume of POSIX.1-2024 affect the behavior of each utility is described in the ENVIRONMENT VARIABLES section for that utility, in conjunction with the global effects of the *LANG*, XSI *LC_ALL*, and *NLSPATH* environment variables described in XBD Chapter 8 (on page 167). The existence or value of environment variables described in this volume of POSIX.1-2024 shall not otherwise affect the specified behavior of the standard utilities. Any effects of the existence or value of environment variables not described by this volume of POSIX.1-2024 upon the standard utilities are unspecified.

For those standard utilities that use environment variables as a means for selecting a utility to execute (such as *CC* in *make*), the string provided to the utility is subjected to the path search described for *PATH* in XBD Chapter 8 (on page 167).

All utilities in this volume of POSIX.1-2024 shall be capable of processing environment variable names and values using eight-bit transparency.

Default Behavior: When this section is listed as ``None.'', it means that the behavior of the utility is not directly affected by environment variables described by this volume of POSIX.1-2024 when the utility is used as described by this volume of POSIX.1-2024.

ASYNCHRONOUS EVENTS

The ASYNCHRONOUS EVENTS section lists how the utility reacts to such events as signals and what signals are caught.

Default Behavior: When this section is listed as ``Default.'', or it refers to ``the standard action'' for any signal, it means that the action taken as a result of the signal shall be as follows:

- If the action inherited from the invoking process, according to the rules of inheritance of signal actions defined in the System Interfaces volume of POSIX.1-2024, is for the signal to be ignored, the utility shall ignore the signal.
- If the action inherited from the invoking process, according to the rules of inheritance of signal actions defined in System Interfaces volume of POSIX.1-2024, is the default signal action, the result of the utility's execution shall be as if the default signal action had been taken.

When the required action is for the signal to terminate the utility, the utility may catch the signal, perform some additional processing (such as deleting temporary files), restore the default signal action, and resignal itself.

STDOUT

The STDOUT section completely describes the standard output of the utility. This section is frequently merely a reference to the following section, OUTPUT FILES, because many utilities treat standard output and output files in the same manner.

Use of a terminal for standard output may cause any of the standard utilities that write standard output to stop when used in the background. For this reason, applications should not use interactive features in scripts to be placed in the background.

Record formats are described in a notation similar to that used by the C-language function, *printf*( ). See XBD Chapter 5 (on page 113) for a description of this notation.

The specified standard output of the standard utilities shall not depend on the existence or value of the environment variables defined in this volume of POSIX.1-2024, except as provided by this volume of POSIX.1-2024.

Some of the standard utilities describe their output using the verb *display*, defined in XBD Section 3.107 (on page 46). Output described in the STDOUT sections of such utilities may be produced using means other than standard output. When standard output is directed to a terminal, the output described shall be written directly to the terminal. Otherwise, the results are undefined.

Default Behavior: When this section is listed as ``Not used.'', it means that the standard output shall not be written when the utility is used as described by this volume of POSIX.1-2024.

STDERR

The STDERR section describes the standard error output of the utility. Only those messages that are purposely sent by the utility are described.

Use of a terminal for standard error may cause any of the standard utilities that write standard error output to stop when used in the background. For this reason, applications should not use interactive features in scripts to be placed in the background.

The format of diagnostic messages for most utilities is unspecified, but the language and cultural conventions of diagnostic and informative messages whose format is unspecified by this volume of POSIX.1-2024 should be affected by the setting of *LC_MESSAGES* and *NLSPATH*. XSI

The specified standard error output of standard utilities shall not depend on the existence or value of the environment variables defined in this volume of POSIX.1-2024, except as provided by this volume of POSIX.1-2024.

Default Behavior: When this section is listed as ``The standard error shall be used only for diagnostic messages.'', it means that, unless otherwise stated, the diagnostic messages shall be sent to the standard error only when the exit status indicates that an error occurred and the utility is used as described by this volume of POSIX.1-2024.

When this section is listed as ``Not used.'', it means that the standard error shall not be used when the utility is used as described in this volume of POSIX.1-2024.

OUTPUT FILES

The OUTPUT FILES section completely describes the files created or modified by the utility. Temporary or system files that are created for internal usage by this utility or other parts of the implementation (for example, spool, log, and audit files) are not described in this, or any, section. The utilities creating such files and the names of such files are unspecified. If applications are written to use temporary or intermediate files, they should use the *TMPDIR* environment variable, if it is set and represents an accessible directory, to select the location of temporary files.

Implementations shall ensure that temporary files, when used by the standard utilities, are named so that different utilities or multiple instances of the same utility can operate simultaneously without regard to their working directories, or any other process characteristic other than process ID. There are two exceptions to this rule:

1. Resources for temporary files other than the name space (for example, disk space, available directory entries, or number of processes allowed) are not guaranteed.
2. Certain standard utilities generate output files that are intended as input for other utilities (for example, *lex* generates **lex.yy.c**), and these cannot have unique names. These cases are explicitly identified in the descriptions of the respective utilities.

Any temporary file created by the implementation shall be removed by the implementation upon a utility's successful exit, exit because of errors, or before termination by any of the SIGHUP, SIGINT, or SIGTERM signals, unless specified otherwise by the utility description.

Receipt of the SIGQUIT signal should generally cause termination (unless in some debugging mode) that would bypass any attempted recovery actions.

Record formats are described in a notation similar to that used by the C-language function, *printf*( ); see XBD Chapter 5 (on page 113) for a description of this notation.

79895 **Default Behavior:** When this section is listed as ``None.'', it means that no files are
79896 created or modified as a consequence of direct action on the part of the utility when the
79897 utility is used as described by this volume of POSIX.1-2024. However, the utility may
79898 create or modify system files, such as log files, that are outside the utility's normal
79899 execution environment.

79900 **EXTENDED DESCRIPTION**

79901 The EXTENDED DESCRIPTION section provides a place for describing the actions of
79902 very complicated utilities, such as text editors or language processors, which typically
79903 have elaborate command languages.

79904 **Default Behavior:** When this section is listed as ``None.'', no further description is
79905 necessary.

79906 **EXIT STATUS**

79907 The EXIT STATUS section describes the values the utility shall return to the calling
79908 program, or shell, and the conditions that cause these values to be returned. Usually,
79909 utilities return zero for successful completion and values greater than zero for various
79910 error conditions. If specific numeric values are listed in this section, the system shall
79911 use those values for the errors described. In some cases, status values are listed more
79912 loosely, such as >0. A strictly conforming application shall not rely on any specific
79913 value in the range shown and shall be prepared to receive any value in the range.

79914 For example, a utility may list zero as a successful return, 1 as a failure for a specific
79915 reason, and >1 as ``an error occurred''. In this case, unspecified conditions may cause a
79916 2 or 3, or other value, to be returned. A conforming application should be written so
79917 that it tests for successful exit status values (zero in this case), rather than relying upon
79918 the single specific error value listed in this volume of POSIX.1-2024. In that way, it has
79919 maximum portability, even on implementations with extensions.

79920 Unspecified error conditions may be represented by specific values not listed in this
79921 volume of POSIX.1-2024.

79922 **Default Behavior:** When the description of exit status 0 is ``Successful completion'', it
79923 means that exit status 0 shall indicate that all of the actions the utility is required to
79924 perform were completed successfully.

79925 **CONSEQUENCES OF ERRORS**

79926 The CONSEQUENCES OF ERRORS section describes the effects on the environment,
79927 file systems, process state, and so on, when error conditions occur. It does not describe
79928 error messages produced or exit status values used.

79929 The many reasons for failure of a utility are generally not specified by the utility
79930 descriptions. Utilities may terminate prematurely if they encounter: invalid usage of
79931 options, arguments, or environment variables; invalid usage of the complex syntaxes
79932 expressed in EXTENDED DESCRIPTION sections; resource exhaustion; difficulties
79933 accessing, creating, reading, or writing files; or difficulties associated with the
79934 privileges of the process.

79935 The following shall apply to each utility, unless otherwise stated:

79936 • If the requested action cannot be performed on an operand representing a file,
79937 directory, user, process, and so on, the utility shall issue a diagnostic message to
79938 standard error and continue processing the next operand in sequence, but the
79939 final exit status shall be one that indicates an error occurred.

79940 For a utility that recursively traverses a file hierarchy (such as *find* or *chown* **–R**), if
79941 the requested action cannot be performed on a file or directory encountered in the

hierarchy, the utility shall issue a diagnostic message to standard error and continue processing the remaining files in the hierarchy, but the final exit status shall be one that indicates an error occurred.

Note: If the requested action is to write one or more pathnames in a format that has <newline> as a terminator or separator, and a pathname to be written contains any bytes that have the encoded value of a <newline> character, this should be treated as an action that cannot be performed. A future version of this standard may require that utilities treat this as an error.

- If the requested action characterized by an option or option-argument cannot be performed, the utility shall issue a diagnostic message to standard error and the exit status returned shall be one that indicates an error occurred.
- When an unrecoverable error condition is encountered, the utility shall exit with an exit status that indicates an error occurred.
- A diagnostic message shall be written to standard error whenever an error condition occurs.

When a utility encounters an error condition several actions are possible, depending on the severity of the error and the state of the utility. Included in the possible actions of various utilities are: deletion of temporary or intermediate work files; deletion of incomplete files; validity checking of the file system or directory.

Default Behavior: When this section is listed as ``Default.'', it means that any changes to the environment, file systems, process state, and so on are unspecified.

APPLICATION USAGE

This section is informative.

The APPLICATION USAGE section gives advice to the application programmer or user about the way the utility should be used.

EXAMPLES

This section is informative.

The EXAMPLES section gives one or more examples of usage, where appropriate. In the event of conflict between an example and a normative part of the specification, the normative material is to be taken as correct.

In all examples, quoting has been used, showing how sample commands (utility names combined with arguments) could be passed correctly to a shell (see *sh*) or as a string to the *system*( ) function defined in the System Interfaces volume of POSIX.1-2024. Such quoting would not be used if the utility is invoked using one of the *exec* functions defined in the System Interfaces volume of POSIX.1-2024.

RATIONALE

This section is informative.

This section contains historical information concerning the contents of this volume of POSIX.1-2024 and why features were included or discarded by the standard developers.

FUTURE DIRECTIONS

This section is informative.

The FUTURE DIRECTIONS section should be used as a guide to current thinking; there is not necessarily a commitment to implement all of these future directions in their entirety.

SEE ALSO

This section is informative.

The SEE ALSO section lists related entries.

CHANGE HISTORY

This section is informative.

This section shows the derivation of the entry and any significant changes that have been made to it.

Certain of the standard utilities describe how they can invoke other utilities or applications, such as by passing a command string to the command interpreter. The external influences (STDIN, ENVIRONMENT VARIABLES, and so on) and external effects (STDOUT, CONSEQUENCES OF ERRORS, and so on) of such invoked utilities are not described in the section concerning the standard utility that invokes them.

1.5 Considerations for Utilities in Support of Files of Arbitrary Size

The following utilities support files of any size up to the maximum that can be created by the implementation. This support includes correct writing of file size-related values (such as file sizes and offsets, line numbers, and block counts) and correct interpretation of command line arguments that contain such values.

basename	Return non-directory portion of pathname.
cat	Concatenate and print files.
cd	Change working directory.
chgrp	Change file group ownership.
chmod	Change file modes.
chown	Change file ownership.
cksum	Write file checksums and sizes.
cmp	Compare two files.
cp	Copy files.
dd	Convert and copy a file.
df	Report free disk space.
dirname	Return directory portion of pathname.
du	Estimate file space usage.
find	Find files.
ln	Link files.
ls	List directory contents.
mkdir	Make directories.
mv	Move files.
pathchk	Check pathnames.

80023 *pwd* Return working directory name.

80024 *rm* Remove directory entries.

80025 *rmdir* Remove directories.

80026 *sh* Shell, the standard command language interpreter.

80027 *test* Evaluate expression.

80028 *touch* Change file access and modification times.

80029 *ulimit* Set or report file size limit.

80030 Exceptions to the requirement that utilities support files of any size up to the maximum are as
80031 follows:

80032 1. Uses of files as command scripts, or for configuration or control, are exempt. For example,
80033 it is not required that *sh* be able to read an arbitrarily large **.profile**.

80034 2. Shell input and output redirection are exempt. For example, it is not required that the
80035 redirections *sum* **<** *file* or *echo foo* **>** *file* succeed for an arbitrarily large existing file.

80036 1.6 Built-In Utilities

80037 Any of the standard utilities may be implemented as regular built-in utilities within the
80038 command language interpreter. This is usually done to increase the performance of frequently
80039 used utilities or to achieve functionality that would be more difficult in a separate environment.
80040 The intrinsic utilities described in Section 1.7 below are frequently provided as regular built-ins.

80041 However, all of the standard utilities other than:

80042 • The special built-ins described in Section 2.15 (on page 2526)

80043 • The intrinsic utilities named in Table 1-5, except for *kill*

80044 shall be implemented, regardless of whether they are also implemented as regular built-ins, in a
80045 manner so that they can be accessed via the *exec* family of functions as defined in the System
80046 Interfaces volume of POSIX.1-2024 and can be invoked directly by those standard utilities that
80047 require it (*env*, *find*, *nice*, *nohup*, *time*, *xargs*).

80048 1.7 Intrinsic Utilities

80049 As described in Section 2.9.1.4 (on page 2502), intrinsic utilities are not subject to a *PATH* search
80050 during command search and execution. The utilities named in Table 1-5 shall be intrinsic
80051 utilities.

80052 **Table 1-5** Intrinsic Utilities

80053 *alias*	*command*	*getopts*	*kill*	*ulimit*	*wait*
80054 *bg*	*fc*	*hash*	*read*	*umask*	
80055 *cd*	*fg*	*jobs*	*type*	*unalias*	

80056 Whether any additional utility is considered an intrinsic utility is implementation-defined.
80057 Because applications are unable to override an intrinsic utility with a utility from *PATH*,
80058 implementations should not make any utility an intrinsic utility beyond the utilities in Table 1-5.

Chapter 2
Shell Command Language

This chapter contains the definition of the Shell Command Language.

2.1 Shell Introduction

The shell is a command language interpreter. This chapter describes the syntax of that command language as it is used by the *sh* utility and the *system*() and *popen*() functions defined in the System Interfaces volume of POSIX.1-2024.

The shell operates according to the following general overview of operations. The specific details are included in the cited sections of this chapter.

1. The shell reads its input from a file (see *sh*), from the **−c** option or from the *system*() and *popen*() functions defined in the System Interfaces volume of POSIX.1-2024. If the first line of a file of shell commands starts with the characters `"#!"`, the results are unspecified.
2. The shell breaks the input into tokens: words and operators; see Section 2.3.
3. The shell parses the input into simple commands (see Section 2.9.1) and compound commands (see Section 2.9.4).
4. For each word within a command, the shell processes <backslash>-escape sequences inside dollar-single-quotes (see Section 2.2.4) and then performs various word expansions (see Section 2.6). In the case of a simple command, the results usually include a list of pathnames and fields to be treated as a command name and arguments; see Section 2.9.
5. The shell performs redirection (see Section 2.7) and removes redirection operators and their operands from the parameter list.
6. The shell executes a function (see Section 2.9.5), built-in (see Section 2.15), executable file, or script, giving the names of the arguments as positional parameters numbered 1 to *n*, and the name of the command (or in the case of a function within a script, the name of the script) as special parameter 0 (see Section 2.9.1.4).
7. The shell optionally waits for the command to complete and collects the exit status (see Section 2.8.2).

2.2 Quoting

Quoting is used to remove the special meaning of certain characters or words to the shell. Quoting can be used to preserve the literal meaning of the special characters in the next paragraph, prevent reserved words from being recognized as such, and prevent parameter expansion and command substitution within here-document processing (see Section 2.7.4).

The application shall quote the following characters if they are to represent themselves:

```
|  &  ;  <  >  (  )  $  `  \  "  '  <space>  <tab>  <newline>
```

and the following might need to be quoted under certain circumstances. That is, these

characters are sometimes special depending on conditions described elsewhere in this volume of POSIX.1-2024:

```
*  ?  [  ]  ^  -  !  #  ~  =  %  {  ,  }
```

Note: A future version of this standard may extend the conditions under which these characters are special. Therefore applications should quote them whenever they are intended to represent themselves. This does not apply to <hyphen-minus> (`'-'`) since it is in the portable filename character set.

The various quoting mechanisms are the escape character, single-quotes, double-quotes, and dollar-single-quotes. The here-document represents another form of quoting; see Section 2.7.4.

2.2.1 Escape Character (Backslash)

A <backslash> that is not quoted shall preserve the literal value of the following character, with the exception of a <newline>. If a <newline> immediately follows the <backslash>, the shell shall interpret this as line continuation. The <backslash> and <newline> shall be removed before splitting the input into tokens. Since the escaped <newline> is removed entirely from the input and is not replaced by any white space, it cannot serve as a token separator.

2.2.2 Single-Quotes

Enclosing characters in single-quotes (`''`) shall preserve the literal value of each character within the single-quotes. A single-quote cannot occur within single-quotes.

2.2.3 Double-Quotes

Enclosing characters in double-quotes (`""`) shall preserve the literal value of all characters within the double-quotes, with the exception of the characters backquote, <dollar-sign>, and <backslash>, as follows:

`$` The <dollar-sign> shall retain its special meaning introducing parameter expansion (see Section 2.6.2), a form of command substitution (see Section 2.6.3), and arithmetic expansion (see Section 2.6.4), but shall not retain its special meaning introducing the dollar-single-quotes form of quoting (see Section 2.2.4).

The input characters within the quoted string that are also enclosed between `"$("` and the matching `')'` shall not be affected by the double-quotes, but rather shall define the command(s) whose output replaces the `"$(...)"` when the word is expanded. The tokenizing rules in Section 2.3 shall be applied recursively to find the matching `')'`.

For the four varieties of parameter expansion that provide for substring processing (see Section 2.6.2), within the string of characters from an enclosed `"${"` to the matching `'}'`, the double-quotes within which the expansion occurs shall have no effect on the handling of any special characters.

For parameter expansions other than the four varieties that provide for substring processing, within the string of characters from an enclosed `"${"` to the matching `'}'`, the double-quotes within which the expansion occurs shall preserve the literal value of all characters, with the exception of the characters double-quote, backquote, <dollar-sign>, and <backslash>. If any unescaped double-quote characters occur within the string, other than in embedded command substitutions, the behavior is unspecified. The backquote and <dollar-sign> characters shall follow the same rules as for characters in double-quotes described in this section. The <backslash> character shall follow the same rules as for

characters in double-quotes described in this section except that it shall additionally retain its special meaning as an escape character when followed by `'}'` and this shall prevent the escaped `'}'` from being considered when determining the matching `'}'` (using the rule in Section 2.6.2).

` The backquote shall retain its special meaning introducing the other form of command substitution (see Section 2.6.3). The portion of the quoted string from the initial backquote and the characters up to the next backquote that is not preceded by a <backslash>, having escape characters removed, defines that command whose output replaces `"`...`"` when the word is expanded. Either of the following cases produces undefined results:

- A quoted (single-quoted, double-quoted, or dollar-single-quoted) string that begins, but does not end, within the `"`...`"` sequence
- A `"`...`"` sequence that begins, but does not end, within the same double-quoted string

\ Outside of `"$(...)"` and `"${...}"` the <backslash> shall retain its special meaning as an escape character (see Section 2.2.1) only when immediately followed by one of the following characters:

```
$    `    \    <newline>
```

or by a double-quote character that would otherwise be considered special (see Section 2.6.4 (on page 2490) and Section 2.7.4, on page 2495).

When double-quotes are used to quote a parameter expansion, command substitution, or arithmetic expansion, the literal value of all characters within the result of the expansion shall be preserved.

The application shall ensure that a double-quote that is not within `"$(...)"` nor within `"${...}"` is immediately preceded by a <backslash> in order to be included within double-quotes. The parameter `'@'` has special meaning inside double-quotes and is described in Section 2.5.2.

2.2.4 Dollar-Single-Quotes

A sequence of characters starting with a <dollar-sign> immediately followed by a single-quote (`$'`) shall preserve the literal value of all characters up to an unescaped terminating single-quote (`'`), with the exception of certain <backslash>-escape sequences, as follows:

- `\"` yields a <quotation-mark> (double-quote) character, but note that <quotation-mark> can be included unescaped.
- `\'` yields an <apostrophe> (single-quote) character.
- `\\` yields a <backslash> character.
- `\a` yields an <alert> character.
- `\b` yields a <backspace> character.
- `\e` yields an <ESC> character.
- `\f` yields a <form-feed> character.
- `\n` yields a <newline> character.
- `\r` yields a <carriage-return> character.

- `\t` yields a <tab> character.
- `\v` yields a <vertical-tab> character.
- `\c`*X* yields the control character listed in the **Value** column of Table 3-21 in the OPERANDS section of the *stty* utility when *X* is one of the characters listed in the **^c** column of the same table, except that `\c\\` yields the <FS> control character since the <backslash> character has to be escaped.
- `\x`*XX* yields the byte whose value is the hexadecimal value *XX* (one or more hexadecimal digits). If more than two hexadecimal digits follow `\x`, the results are unspecified.
- `\`*ddd* yields the byte whose value is the octal value *ddd* (one to three octal digits).
- The behavior of an unescaped <backslash> immediately followed by any other character, including <newline>, is unspecified.

In cases where a variable number of characters can be used to specify an escape sequence (`\x`*XX* and `\`*ddd*), the escape sequence shall be terminated by the first character that is not of the expected type or, for `\`*ddd* sequences, when the maximum number of characters specified has been found, whichever occurs first.

These <backslash>-escape sequences shall be processed (replaced with the bytes or characters they yield) immediately prior to word expansion (see Section 2.6) of the word in which the dollar-single-quotes sequence occurs.

If a `\x`*XX* or `\`*ddd* escape sequence yields a byte whose value is 0, it is unspecified whether that null byte is included in the result or if that byte and any following regular characters and escape sequences up to the terminating unescaped single-quote are evaluated and discarded.

If the octal value specified by `\`*ddd* will not fit in a byte, the results are unspecified.

If a `\e` or `\c`*X* escape sequence specifies a character that does not have an encoding in the locale in effect when these <backslash>-escape sequences are processed, the result is implementation-defined. However, implementations shall not replace an unsupported character with bytes that do not form valid characters in that locale's character set.

If a <backslash>-escape sequence represents a single-quote character (for example `\'`), that sequence shall not terminate the dollar-single-quote sequence.

2.3 Token Recognition

The shell shall read its input in terms of lines. (For details about how the shell reads its input, see the description of *sh*.) The input lines can be of unlimited length. These lines shall be parsed using two major modes: ordinary token recognition and processing of here-documents.

When an **io_here** token has been recognized by the grammar (see Section 2.10), one or more of the subsequent lines immediately following the next **NEWLINE** token form the body of a here-document and shall be parsed according to the rules of Section 2.7.4. Any non-**NEWLINE** tokens (including more **io_here** tokens) that are recognized while searching for the next **NEWLINE** token shall be saved for processing after the here-document has been parsed. If a saved token is an **io_here** token, the corresponding here-document shall start on the line immediately following the line containing the trailing delimiter of the previous here-document. If any saved token includes a <newline> character, the behavior is unspecified.

When it is not processing an **io_here**, the shell shall break its input into tokens by applying the first applicable rule below to each character in turn in its input. At the start of input or after a previous token has just been delimited, the first or next token, respectively, shall start with the first character that has not already been included in a token and is not discarded according to

the rules below. Once a token has started, zero or more characters from the input shall be appended to the token until the end of the token is delimited according to one of the rules below. When both the start and end of a token have been delimited, the characters forming the token shall be exactly those in the input between the two delimiters, including any quoting characters. If a rule below indicates that a token is delimited, and no characters have been included in the token, that empty token shall be discarded.

1. If the end of input is recognized, the current token (if any) shall be delimited.
2. If the previous character was used as part of an operator and the current character is not quoted and can be used with the previous characters to form an operator, it shall be used as part of that (operator) token.
3. If the previous character was used as part of an operator and the current character cannot be used with the previous characters to form an operator, the operator containing the previous character shall be delimited.
4. If the current character is an unquoted <backslash>, single-quote, or double-quote or is the first character of an unquoted <dollar-sign> single-quote sequence, it shall affect quoting for subsequent characters up to the end of the quoted text. The rules for quoting are as described in Section 2.2. During token recognition no substitutions shall be actually performed, and the result token shall contain exactly the characters that appear in the input unmodified, including any embedded or enclosing quotes or substitution operators, between the start and the end of the quoted text. The token shall not be delimited by the end of the quoted field.
5. If the current character is an unquoted `'$'` or `'`'`, the shell shall identify the start of any candidates for parameter expansion (Section 2.6.2), command substitution (Section 2.6.3), or arithmetic expansion (Section 2.6.4) from their introductory unquoted character sequences: `'$'` or `"${"`, `"$("` or `'`'`, and `"$(("`, respectively. The shell shall read sufficient input to determine the end of the unit to be expanded (as explained in the cited sections). While processing the characters, if instances of expansions or quoting are found nested within the substitution, the shell shall recursively process them in the manner specified for the construct that is found. For `"$("` and `'`'` only, if instances of **io_here** tokens are found nested within the substitution, they shall be parsed according to the rules of Section 2.7.4; if the terminating `')'` or `'`'` of the substitution occurs before the **NEWLINE** token marking the start of the here-document, the behavior is unspecified. The characters found from the beginning of the substitution to its end, allowing for any recursion necessary to recognize embedded constructs, shall be included unmodified in the result token, including any embedded or enclosing substitution operators or quotes. The token shall not be delimited by the end of the substitution.
6. If the current character is not quoted and can be used as the first character of a new operator, the current token (if any) shall be delimited. The current character shall be used as the beginning of the next (operator) token.
7. If the current character is an unquoted <blank>, any token containing the previous character is delimited and the current character shall be discarded.
8. If the previous character was part of a word, the current character shall be appended to that word.
9. If the current character is a `'#'`, it and all subsequent characters up to, but excluding, the next <newline> shall be discarded as a comment. The <newline> that ends the line is not considered part of the comment.

10. The current character is used as the start of a new word.

Once a token is delimited, it is categorized as required by the grammar in Section 2.10.

In situations where the shell parses its input as a *program*, once a *complete_command* has been recognized by the grammar (see Section 2.10), the *complete_command* shall be executed before the next *complete_command* is tokenized and parsed.

2.3.1 Alias Substitution

After a token has been categorized as type **TOKEN** (see Section 2.10.1), including (recursively) any token resulting from an alias substitution, the **TOKEN** shall be subject to alias substitution if all of the following conditions are true:

- The **TOKEN** does not contain any quoting characters.
- The **TOKEN** is a valid alias name (see XBD Section 3.10).
- An alias with that name is in effect.
- The **TOKEN** did not either fully or, optionally, partially result from an alias substitution of the same alias name at any earlier recursion level.
- Either the **TOKEN** is being considered for alias substitution because it follows an alias substitution whose replacement value ended with a <blank> (see below) or the **TOKEN** could be parsed as the command name word of a simple command (see Section 2.10), based on this **TOKEN** and the tokens (if any) that preceded it, but ignoring whether any subsequent characters would allow that.

except that if the **TOKEN** meets the above conditions and would be recognized as a reserved word (see Section 2.4) if it occurred in an appropriate place in the input, it is unspecified whether the **TOKEN** is subject to alias substitution.

When a **TOKEN** is subject to alias substitution, the value of the alias shall be processed as if it had been read from the input instead of the **TOKEN**, with token recognition (see Section 2.3) resuming at the start of the alias value. When the end of the alias value is reached, the shell may behave as if an additional <space> character had been read from the input after the **TOKEN** that was replaced. If it does not add this <space>, it is unspecified whether the current token is delimited before token recognition is applied to the character (if any) that followed the **TOKEN** in the input.

Note: A future version of this standard may disallow adding this <space>.

If the value of the alias replacing the **TOKEN** ends in a <blank> that would be unquoted after substitution, and optionally if it ends in a <blank> that would be quoted after substitution, the shell shall check the next token in the input, if it is a **TOKEN**, for alias substitution; this process shall continue until a **TOKEN** is found that is not a valid alias or an alias value does not end in such a <blank>.

An implementation may defer the effect of a change to an alias but the change shall take effect no later than the completion of the currently executing *complete_command* (see Section 2.10). Changes to aliases shall not take effect out of order. Implementations may provide predefined aliases that are in effect when the shell is invoked.

When used as specified by this volume of POSIX.1-2024, alias definitions shall not be inherited by separate invocations of the shell or by the utility execution environments invoked by the shell; see Section 2.13.

2.4 Reserved Words

Reserved words are words that have special meaning to the shell; see Section 2.9. The following words shall be recognized as reserved words:

!	**do**	**esac**	**in**
{	**done**	**fi**	**then**
}	**elif**	**for**	**until**
case	**else**	**if**	**while**

This recognition shall only occur when none of the characters is quoted and when the word is used as:

- The first word of a command
- The first word following one of the reserved words other than **case**, **for**, or **in**
- The third word in a **case** command (only **in** is valid in this case)
- The third word in a **for** command (only **in** and **do** are valid in this case)

See the grammar in Section 2.10.

When used in circumstances where reserved words are recognized (described above), the following words may be recognized as reserved words, in which case the results are unspecified except as described below for **time**:

[[	**]]**	**function**	**namespace**	**select**	**time**

When the word **time** is recognized as a reserved word in circumstances where it would, if it were not a reserved word, be the command name (see Section 2.9.1.1) of a simple command that would execute the *time* utility in a manner other than one for which *time* states that the results are unspecified, the behavior shall be as specified for the *time* utility.

When used in circumstances where reserved words are recognized (described above), all words whose final character is a <colon> (`':'`) are reserved; their use in those circumstances produces unspecified results.

2.5 Parameters and Variables

A parameter can be denoted by a name, a number, or one of the special characters listed in Section 2.5.2. A variable is a parameter denoted by a name.

A parameter is set if it has an assigned value (null is a valid value). Once a variable is set, it can only be unset by using the *unset* special built-in command.

Parameters can contain arbitrary byte sequences, except for the null byte. The shell shall process their values as characters only when performing operations that are described in this standard in terms of characters.

2.5.1 Positional Parameters

A positional parameter is a parameter denoted by a decimal representation of a positive integer. The digits denoting the positional parameters shall always be interpreted as a decimal value, even if there is a leading zero. When a positional parameter with more than one digit is specified, the application shall enclose the digits in braces (see Section 2.6.2).

Examples:

- `"$8"`, `"${8}"`, `"${08}"`, `"${008}"`, etc. all expand to the value of the eighth positional parameter.
- `"${10}"` expands to the value of the tenth positional parameter.
- `"$10"` expands to the value of the first positional parameter followed by the character '0'.

Note: 0 is a special parameter, not a positional parameter, and therefore the results of expanding `${00}` are unspecified.

Positional parameters are initially assigned when the shell is invoked (see *sh*), temporarily replaced when a shell function is invoked (see Section 2.9.5), and can be reassigned with the *set* special built-in command.

2.5.2 Special Parameters

Listed below are the special parameters and the values to which they shall expand. Only the values of the special parameters are listed; see Section 2.6 for a detailed summary of all the stages involved in expanding words.

`@` Expands to the positional parameters, starting from one, initially producing one field for each positional parameter that is set. When the expansion occurs in a context where field splitting will be performed, any empty fields may be discarded and each of the non-empty fields shall be further split as described in Section 2.6.5. When the expansion occurs within double-quotes, the behavior is unspecified unless one of the following is true:

- Field splitting as described in Section 2.6.5 would be performed if the expansion were not within double-quotes (regardless of whether field splitting would have any effect; for example, if *IFS* is null).
- The double-quotes are within the *word* of a ${*parameter*:–*word*} or a ${*parameter*:+*word*} expansion (with or without the <colon>; see Section 2.6.2) which would have been subject to field splitting if *parameter* had been expanded instead of *word*.

If one of these conditions is true, the initial fields shall be retained as separate fields, except that if the parameter being expanded was embedded within a word, the first field shall be joined with the beginning part of the original word and the last field shall be joined with the end part of the original word. In all other contexts the results of the expansion are unspecified. If there are no positional parameters, the expansion of `'@'` shall generate zero fields, even when `'@'` is within double-quotes; however, if the expansion is embedded within a word which contains one or more other parts that expand to a quoted null string, these null string(s) shall still produce an empty field, except that if the other parts are all within the same double-quotes as the `'@'`, it is unspecified whether the result is zero fields or one empty field.

`*` Expands to the positional parameters, starting from one, initially producing one field for each positional parameter that is set. When the expansion occurs in a context where field splitting will be performed, any empty fields may be discarded and each of the non-empty fields shall be further split as described in Section 2.6.5. When the expansion occurs in a

80386 context where field splitting will not be performed, the initial fields shall be joined to form a
80387 single field with the value of each parameter separated by the first character of the *IFS*
80388 variable if *IFS* contains at least one character, or separated by a <space> if *IFS* is unset, or
80389 with no separation if *IFS* is set to a null string.

80390 # Expands to the shortest representation of the decimal number of positional parameters. The
80391 command name (parameter 0) shall not be counted in the number given by `'#'` because it is
80392 a special parameter, not a positional parameter.

80393 ? Expands to the shortest representation of the decimal exit status (see Section 2.8.2) of the
80394 pipeline (see Section 2.9.2) executed from the current shell execution environment (not a
80395 subshell environment) that most recently either terminated or, optionally but only if the
80396 shell is interactive and job control is enabled, was stopped by a signal. If this pipeline
80397 terminated, the status value shall be its exit status; otherwise, the status value shall be the
80398 same as the exit status that would have resulted if the pipeline had been terminated by a
80399 signal with the same number as the signal that stopped it. The value of the special
80400 parameter `'?'` shall be set to 0 during initialization of the shell. When a subshell
80401 environment is created, the value of the special parameter `'?'` from the invoking shell
80402 environment shall be preserved in the subshell.

80403 **Note:** In `var=$(some_command); echo $?` the output is the exit status of `some_command`,
80404 which is executed in a subshell environment, but this is because its exit status becomes the
80405 exit status of the assignment command `var=$(some_command)` (see Section 2.9.1) and
80406 this assignment command is the most recently completed pipeline. Likewise for any
80407 pipeline consisting entirely of a simple command that has no command word, but contains
80408 one or more command substitutions. (See Section 2.9.1.)

80409 – (Hyphen.) Expands to the current option flags (the single-letter option names concatenated
80410 into a string) as specified on invocation, by the *set* special built-in command, or implicitly
80411 by the shell. It is unspecified whether the **–c** and **–s** options are included in the expansion
80412 of `"$-"`. The **–i** option shall be included in `"$-"` if the shell is interactive, regardless of
80413 whether it was specified on invocation.

80414 $ Expands to the shortest representation of the decimal process ID of the invoked shell. In a
80415 subshell (see Section 2.13), `'$'` shall expand to the same value as that of the current shell.

80416 ! Expands to the shortest representation of the decimal process ID associated with the most
80417 recent asynchronous AND-OR list (see Section 2.9.3.1) executed from the current shell
80418 execution environment, or to the shortest representation of the decimal process ID of the
80419 last command specified in the currently executing pipeline in the job-control background
80420 job that most recently resumed execution through the use of *bg*, whichever is the most
80421 recent.

80422 0 (Zero.) Expands to the name of the shell or shell script. See *sh* for a detailed description of
80423 how this name is derived.

80424 See the description of the *IFS* variable in Section 2.5.3.

2.5.3 Shell Variables

Variables shall be initialized from the environment (as defined by XBD Chapter 8 and the *exec* function in the System Interfaces volume of POSIX.1-2024) and can be given new values with variable assignment commands. Shell variables shall be initialized only from environment variables that have valid names. If a variable is initialized from the environment, it shall be marked for export immediately; see the *export* special built-in. New variables can be defined and initialized with variable assignments, with the *read* or *getopts* utilities, with the *name* parameter in a **for** loop, with the ${*name*=*word*} expansion, or with other mechanisms provided as implementation extensions.

The following variables shall affect the execution of the shell:

UP *ENV* The processing of the *ENV* shell variable shall be supported if the system supports the User Portability Utilities option.

This variable, when and only when an interactive shell is invoked, shall be subjected to parameter expansion (see Section 2.6.2) by the shell and the resulting value shall be used as a pathname of a file. Before any interactive commands are read, the shell shall tokenize (see Section 2.3) the contents of the file, parse the tokens as a *program* (see Section 2.10), and execute the resulting commands in the current environment. (In other words, the contents of the *ENV* file are not parsed as a single *compound_list*. This distinction matters because it influences when aliases take effect.) The file need not be executable. If the expanded value of *ENV* is not an absolute pathname, the results are unspecified. *ENV* shall be ignored if the user's real and effective user IDs or real and effective group IDs are different.

HOME The pathname of the user's home directory. The contents of *HOME* are used in tilde expansion (see Section 2.6.1).

IFS A string treated as a list of characters that is used for field splitting, expansion of the `'*'` special parameter, and to split lines into fields with the *read* utility. If the value of *IFS* includes any bytes that do not form part of a valid character, the results of field splitting, expansion of `'*'`, and use of the *read* utility are unspecified.

If *IFS* is not set, it shall behave as normal for an unset variable, except that field splitting by the shell and line splitting by the *read* utility shall be performed as if the value of *IFS* is <space><tab><newline>; see Section 2.6.5.

The shell shall set *IFS* to <space><tab><newline> when it is invoked.

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL The value of this variable overrides the *LC_** variables and *LANG*, as described in XBD Chapter 8.

LC_COLLATE Determine the behavior of range expressions, equivalence classes, and multi-character collating elements within pattern matching.

LC_CTYPE Determine the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters), which characters are defined as letters (character class **alpha**) and <blank> characters (character class **blank**), and the behavior of character classes within pattern matching. Changing the value of *LC_CTYPE* after the shell has started shall not affect the lexical processing of shell commands in the current shell

execution environment or its subshells. Invoking a shell script or performing *exec sh* subjects the new shell to the changes in *LC_CTYPE*.

LC_MESSAGES Determine the language in which messages should be written.

UP *LINENO* The processing of the *LINENO* shell variable shall be supported if the system supports the User Portability Utilities option.

Set by the shell to a decimal number representing the current sequential line number (numbered starting with 1) within a script or function before it executes each command. If the user unsets or resets *LINENO*, the variable may lose its special meaning for the life of the shell. If the shell is not currently executing a script or function, the value of *LINENO* is unspecified.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

PATH A string formatted as described in XBD Chapter 8, used to effect command interpretation; see Section 2.9.1.4.

PPID Set by the shell to the decimal value of its parent process ID during initialization of the shell. In a subshell (see Section 2.13), *PPID* shall be set to the same value as that of the parent of the current shell. For example, *echo* $*PPID* and (*echo* $*PPID*) would produce the same value.

UP *PS1* The processing of the *PS1* shell variable shall be supported if the system supports the User Portability Utilities option.

Each time an interactive shell is ready to read a command, the value of this variable shall be subjected to parameter expansion (see Section 2.6.2) and exclamation-mark expansion (see below). Whether the value is also subjected to command substitution (see Section 2.6.3) or arithmetic expansion (see Section 2.6.4) or both is unspecified. After expansion, the value shall be written to standard error.

The expansions shall be performed in two passes, where the result of the first pass is input to the second pass. One of the passes shall perform only the exclamation-mark expansion described below. The other pass shall perform the other expansion(s) according to the rules in Section 2.6. Which of the two passes is performed first is unspecified.

The default value shall be `"$ "`. For users who have specific additional implementation-defined privileges, the default may be another, implementation-defined value.

Exclamation-mark expansion: The shell shall replace each instance of the <exclamation-mark> character (`'!'`) with the history file number (see Command History List) of the next command to be typed. An <exclamation-mark> character escaped by another <exclamation-mark> character (that is, `"!!"`) shall expand to a single <exclamation-mark> character.

UP *PS2* The processing of the *PS2* shell variable shall be supported if the system supports the User Portability Utilities option.

Each time the user enters a <newline> prior to completing a command line in an interactive shell, the value of this variable shall be subjected to parameter expansion (see Section 2.6.2). Whether the value is also subjected to command substitution (see Section 2.6.3) or arithmetic expansion (see Section 2.6.4) or both is unspecified. After expansion, the value shall be written to standard

error. The default value shall be "> ".

UP *PS4* The processing of the *PS4* shell variable shall be supported if the system supports the User Portability Utilities option.

When an execution trace (*set* –**x**) is being performed, before each line in the execution trace, the value of this variable shall be subjected to parameter expansion (see Section 2.6.2). Whether the value is also subjected to command substitution (see Section 2.6.3) or arithmetic expansion (see Section 2.6.4) or both is unspecified. After expansion, the value shall be written to standard error. The default value shall be "+ ".

PWD Set by the shell and by the *cd* utility. In the shell the value shall be initialized from the environment as follows. If a value for *PWD* is passed to the shell in the environment when it is executed, the value is an absolute pathname of the current working directory that is no longer than {PATH_MAX} bytes including the terminating null byte, and the value does not contain any components that are dot or dot-dot, then the shell shall set *PWD* to the value from the environment. Otherwise, if a value for *PWD* is passed to the shell in the environment when it is executed, the value is an absolute pathname of the current working directory, and the value does not contain any components that are dot or dot-dot, then it is unspecified whether the shell sets *PWD* to the value from the environment or sets *PWD* to the pathname that would be output by *pwd* –**P**. Otherwise, the *sh* utility sets *PWD* to the pathname that would be output by *pwd* –**P**. In cases where *PWD* is set to the value from the environment, the value can contain components that refer to files of type symbolic link. In cases where *PWD* is set to the pathname that would be output by *pwd* –**P**, if there is insufficient permission on the current working directory, or on any parent of that directory, to determine what that pathname would be, the value of *PWD* is unspecified. Assignments to this variable may be ignored. If an application sets or unsets the value of *PWD*, the behaviors of the *cd* and *pwd* utilities are unspecified.

2.6 Word Expansions

This section describes the various expansions that are performed on words. Not all expansions are performed on every word, as explained in the following sections and elsewhere in this chapter. The expansions that are performed for a given word shall be performed in the following order:

1. Tilde expansion (see Section 2.6.1), parameter expansion (see Section 2.6.2), command substitution (see Section 2.6.3), and arithmetic expansion (see Section 2.6.4) shall be performed, beginning to end. See item 5 in Section 2.3.
2. Field splitting (see Section 2.6.5) shall be performed on the portions of the fields generated by step 1.
3. Pathname expansion (see Section 2.6.6) shall be performed, unless *set* –**f** is in effect.
4. Quote removal (see Section 2.6.7), if performed, shall always be performed last.

Tilde expansions, parameter expansions, command substitutions, arithmetic expansions, and quote removals that occur within a single word shall expand to a single field, except as described below. The shell shall create multiple fields or no fields from a single word only as a result of field splitting, pathname expansion, or the following cases:

80563 1. Parameter expansion of the special parameters `'@'` and `'*'`, as described in Section 2.5.2,
80564 can create multiple fields or no fields from a single word.

80565 2. When the expansion occurs in a context where field splitting will be performed, a word
80566 that contains all of the following somewhere within it, before any expansions are applied,
80567 in the order specified:

80568 • an unquoted <left-curly-bracket> (`'{'`) that is not immediately preceded by an
80569 unquoted <dollar-sign> (`'$'`)

80570 • one or more unquoted <comma> (`','`) characters or a sequence that consists of two
80571 adjacent <period> (`'.'`) characters surrounded by other characters (which can also
80572 be <period> characters)

80573 • an unquoted <right-curly-bracket> (`'}'`)

80574 may be subject to an additional implementation-defined form of expansion that can create
80575 multiple fields from a single word. This expansion, if supported, shall be applied before
80576 all the other word expansions are applied. The other expansions shall then be applied to
80577 each field that results from this expansion.

80578 When the expansions in this section are performed other than in the context of preparing a
80579 command for execution, they shall be carried out in the current shell execution environment.

80580 When expanding words for a command about to be executed, and the word will be the
80581 command name or an argument to the command, the expansions shall be carried out in the
80582 current shell execution environment. (The environment for the command to be executed is
80583 unknown until the command word is known.)

80584 When expanding the words in a command about to be executed that are used with variable
80585 assignments or redirections, it is unspecified whether the expansions are carried out in the
80586 current execution environment or in the environment of the command about to be executed.

80587 The `'$'` character is used to introduce parameter expansion, command substitution, or
80588 arithmetic evaluation. If a `'$'` that is neither within single-quotes nor escaped by a <backslash>
80589 is immediately followed by a character that is not a <space>, not a <tab>, not a <newline>, and
80590 is not one of the following:

80591 • A numeric character

80592 • The name of one of the special parameters (see Section 2.5.2)

80593 • A valid first character of a variable name

80594 • A <left-curly-bracket> (`'{'`)

80595 • A <left-parenthesis>

80596 • A single-quote

80597 the result is unspecified. If a `'$'` that is neither within single-quotes nor escaped by a
80598 <backslash> is immediately followed by a <space>, <tab>, or a <newline>, or is not followed by
80599 any character, the `'$'` shall be treated as a literal character.

2.6.1 Tilde Expansion

A ``tilde-prefix'' consists of an unquoted <tilde> character at the beginning of a word, followed by all of the characters preceding the first unquoted <slash> in the word, or all the characters in the word if there is no <slash>. In an assignment (see XBD Section 4.26), multiple tilde-prefixes can be used: one at the beginning of the word (that is, following the <equals-sign> of the assignment), or one following any unquoted <colon>, or both. A tilde-prefix in an assignment is terminated by the first unquoted <colon> or <slash>, or the end of the assignment word.

If the tilde-prefix consists of only the <tilde> character, it shall be replaced by the value of the variable *HOME*. If *HOME* is unset, the results are unspecified.

Otherwise, the characters in the tilde-prefix following the <tilde> shall be treated as a possible login name from the user database. If these characters do not form a portable login name (see the description of the *LOGNAME* environment variable in XBD Section 8.3), the results are unspecified.

Note: Since the tilde-prefix is not subject to further word expansions after the <tilde> is removed to obtain the login name, none of the following has a portable login name following the <tilde>:

```
~"string"
~'string'
~$var
~\/bin
```

owing to the presence of `'"'`, `'\''`, `'$'`, `'\\'`, and `'/'` characters in the login name.

If the characters in the tilde-prefix following the <tilde> form a portable login name, the tilde-prefix shall be replaced by a pathname of the initial working directory associated with the login name. The pathname shall be obtained as if by using the *getpwnam*( ) function as defined in the System Interfaces volume of POSIX.1-2024. If the system does not recognize the login name, the results are unspecified.

The pathname that replaces the tilde-prefix shall be treated as if quoted to prevent it being altered by field splitting and pathname expansion; if a <slash> follows the tilde-prefix and the pathname ends with a <slash>, the trailing <slash> from the pathname should be omitted from the replacement. If the word being expanded consists of only the <tilde> character and *HOME* is set to the null string, this produces an empty field (as opposed to zero fields) as the expanded word.

Note: A future version of this standard may require that if a <slash> follows the tilde-prefix and the pathname ends with a <slash>, the trailing <slash> from the pathname is omitted from the replacement.

2.6.2 Parameter Expansion

The format for parameter expansion is as follows:

`${`*`expression`*`}`

where *expression* consists of all characters until the matching `'}'`. Any `'}'` escaped by a <backslash> or within a quoted string, and characters in embedded arithmetic expansions, command substitutions, and variable expansions, shall not be examined in determining the matching `'}'`.

The simplest form for parameter expansion is:

`${`*`parameter`*`}`

The value, if any, of *parameter* shall be substituted.

The parameter name or symbol can be enclosed in braces, which are optional except for positional parameters with more than one digit or when *parameter* is a name and is followed by a character that could be interpreted as part of the name.

For a parameter that is not enclosed in braces:

- If the parameter is a name, the expansion shall use the longest valid name (see XBD Section 3.216), whether or not the variable denoted by that name exists.
- Otherwise, the parameter is a single-character symbol, and behavior is unspecified if that character is neither a digit nor one of the special parameters (see Section 2.5.2).

In addition, a parameter expansion can be modified by using one of the following formats. In each case that a value of *word* is needed (based on the state of *parameter*, as described below), *word* shall be subjected to tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote removal. If *word* is not needed, it shall not be expanded. The `'}'` character that delimits the following parameter expansion modifications shall be determined as described previously in this section and in Section 2.2.3. If *parameter* is `'*'` or `'@'`, the result of the expansion is unspecified.

${*parameter*:–[*word*]} **Use Default Values**. If *parameter* is unset or null, the expansion of *word* (or an empty string if *word* is omitted) shall be substituted; otherwise, the value of *parameter* shall be substituted.

${*parameter*:=[*word*]} **Assign Default Values**. If *parameter* is unset or null, quote removal shall be performed on the expansion of *word* and the result (or an empty string if *word* is omitted) shall be assigned to *parameter*. In all cases, the final value of *parameter* shall be substituted. Only variables, not positional parameters or special parameters, can be assigned in this way.

${*parameter*:?[*word*]} **Indicate Error if Null or Unset**. If *parameter* is unset or null, the expansion of *word* (or a message indicating it is unset if *word* is omitted) shall be written to standard error and the shell exits with a non-zero exit status. Otherwise, the value of *parameter* shall be substituted. An interactive shell need not exit.

${*parameter*:+[*word*]} **Use Alternative Value**. If *parameter* is unset or null, null shall be substituted; otherwise, the expansion of *word* (or an empty string if *word* is omitted) shall be substituted.

In the parameter expansions shown previously, use of the <colon> in the format shall result in a test for a parameter that is unset or null; omission of the <colon> shall result in a test for a parameter that is only unset. If parameter is `'#'` and the colon is omitted, the application shall ensure that *word* is specified (this is necessary to avoid ambiguity with the string length expansion). The following table summarizes the effect of the <colon>:

	parameter **Set and Not Null**	*parameter* **Set But Null**	*parameter* **Unset**
${*parameter***:–***word***}**	substitute *parameter*	substitute *word*	substitute *word*
${*parameter***–***word***}**	substitute *parameter*	substitute null	substitute *word*
${*parameter***:=***word***}**	substitute *parameter*	assign *word*	assign *word*
${*parameter***=***word***}**	substitute *parameter*	substitute null	assign *word*
${*parameter***:?***word***}**	substitute *parameter*	error, exit	error, exit
${*parameter***?***word***}**	substitute *parameter*	substitute null	error, exit
${*parameter***:+***word***}**	substitute *word*	substitute null	substitute null
${*parameter***+***word***}**	substitute *word*	substitute *word*	substitute null

In all cases shown with ``substitute'', the expression is replaced with the value shown. In all cases shown with ``assign'', *parameter* is assigned that value, which also replaces the expression.

${#*parameter*} **String Length**. The shortest decimal representation of the length in characters of the value of *parameter* shall be substituted. If *parameter* is `'*'` or `'@'`, the result of the expansion is unspecified. If *parameter* is unset and *set* –**u** is in effect, the expansion shall fail.

The following four varieties of parameter expansion provide for character substring processing. In each case, pattern matching notation (see Section 2.14), rather than regular expression notation, shall be used to evaluate the patterns. If *parameter* is `'#'`, `'*'`, or `'@'`, the result of the expansion is unspecified. If *parameter* is unset and *set* –**u** is in effect, the expansion shall fail. Enclosing the full parameter expansion string in double-quotes shall not cause the following four varieties of pattern characters to be quoted, whereas quoting characters within the braces shall have this effect. In each variety, if *word* is omitted, the empty pattern shall be used.

${*parameter*%**[***word***]**} **Remove Smallest Suffix Pattern**. The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the smallest portion of the suffix matched by the *pattern* deleted. If present, *word* shall not begin with an unquoted `'%'`.

${*parameter*%%**[***word***]**} **Remove Largest Suffix Pattern**. The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the largest portion of the suffix matched by the *pattern* deleted.

${*parameter*#**[***word***]**} **Remove Smallest Prefix Pattern**. The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the smallest portion of the prefix matched by the *pattern* deleted. If present, *word* shall not begin with an unquoted `'#'`.

${*parameter*##**[***word***]**} **Remove Largest Prefix Pattern**. The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the largest portion of the prefix matched by the *pattern* deleted.

Examples

${*parameter*}

In this example, the effects of omitting braces are demonstrated.

```
a=1
set 2
echo ${a}b-$ab-${1}0-${10}-$10
1b--20--20
```


${*parameter–word*}
This example demonstrates the difference between unset and set to the empty string, as well as the rules for finding the delimiting close brace.

```
foo=asdf
echo ${foo-bar}xyz}
asdfxyz}
foo=
echo ${foo-bar}xyz}
xyz}
unset foo
echo ${foo-bar}xyz}
barxyz}
```

${*parameter*:–*word*}
In this example, *ls* is executed only if *x* is null or unset. (The $(*ls*) command substitution notation is explained in Section 2.6.3.)

```
${x:-$(ls)}
```

${*parameter*:=*word*}

```
unset X
echo ${X:=abc}
abc
```

${*parameter*:?*word*}

```
unset posix
echo ${posix:?}
sh: posix: parameter null or not set
```

${*parameter*:+*word*}

```
set a b c
echo ${3:+posix}
posix
```

${#*parameter*}

```
HOME=/usr/posix
echo ${#HOME}
10
```

${*parameter*%*word*}

```
x=file.c
echo ${x%.c}.o
file.o
```

${*parameter*%%*word*}

```
x=posix/src/std
echo ${x%%/*}
posix
```

${*parameter*#*word*}

```
x=$HOME/src/cmd
echo ${x#$HOME}
/src/cmd
```


80768 ${*parameter*##*word*}
80769 `x=/one/two/three`
80770 `echo ${x##*/}`
80771 **three**

80772 The double-quoting of patterns is different depending on where the double-quotes are placed:

80773 `"${x#*}"` The <asterisk> is a pattern character.

80774 `${x#"*"}` The literal <asterisk> is quoted and not special.

80775 ### 2.6.3 Command Substitution

80776 Command substitution allows the output of one or more commands to be substituted in place of
80777 the commands themselves. Command substitution shall occur when command(s) are enclosed
80778 as follows:

80779 `$(`*commands*`)`

80780 or (backquoted version):

80781 `` `commands` ``

80782 The shell shall expand the command substitution by executing *commands* in a subshell
80783 environment (see Section 2.13) and replacing the command substitution (the text of the
80784 *commands* string plus the enclosing `"$()"` or backquotes) with the standard output of the
80785 command(s); if the output ends with one or more bytes that have the encoded value of a
80786 <newline> character, they shall not be included in the replacement. Any such bytes that occur
80787 elsewhere shall be included in the replacement; however, they might be treated as field
80788 delimiters and eliminated during field splitting, depending on the value of *IFS* and quoting that
80789 is in effect. If the output contains any null bytes, the behavior is unspecified.

80790 Within the backquoted style of command substitution, if the command substitution is not within
80791 double-quotes, <backslash> shall retain its literal meaning, except when followed by: `'$'`, `'`'`,
80792 or <backslash>. See Section 2.2.3 for the handling of <backslash> when the command
80793 substitution is within double-quotes. The search for the matching backquote shall be satisfied
80794 by the first unquoted non-escaped backquote; during this search, if a non-escaped backquote is
80795 encountered within a shell comment, a here-document, an embedded command substitution of
80796 the $(*commands*) form, or a quoted string, undefined results occur. A quoted string that begins,
80797 but does not end, within the `"`...`"` sequence produces undefined results.

80798 With the $(*commands*) form, all characters following the open parenthesis to the matching closing
80799 parenthesis constitute the *commands* string.

80800 With both the backquoted and $(*commands*) forms, the *commands* string shall be tokenized (see
80801 Section 2.3) and parsed (see Section 2.10). It is unspecified whether the *commands* string is
80802 parsed and executed incrementally as a *program* (as for a shell script), or is parsed as a single
80803 *compound_list* that is executed after the string has been completely parsed. In addition, it is
80804 unspecified whether the terminating `')'` of the $(*commands*) form can result from alias
80805 substitution. With the $(*commands*) form any syntactically correct *program* can be used for
80806 *commands*, except that:

80807 - If the *commands* string consists solely of redirections, the results are unspecified.

80808 - If the *commands* string is parsed as a single *compound_list*, before any commands are
80809 executed, *alias* and *unalias* commands in *commands* have no effect during parsing (see
80810 Section 2.3.1). Strictly conforming applications shall ensure that the *commands* string does

not depend on alias changes taking effect incrementally as would be the case if parsed and executed as a *program*.

- The behavior is unspecified if the terminating `')'` is not present in the token containing the command substitution; that is, if the `')'` is expected to result from alias substitution.

The results of command substitution shall not be processed for further tilde expansion, parameter expansion, command substitution, or arithmetic expansion.

Command substitution can be nested. To specify nesting within the backquoted version, the application shall precede the inner backquotes with <backslash> characters; for example:

```
\`commands\`
```

The syntax of the shell command language has an ambiguity for expansions beginning with `"$(("`, which can introduce an arithmetic expansion or a command substitution that starts with a subshell. Arithmetic expansion has precedence; that is, the shell shall first determine whether it can parse the expansion as an arithmetic expansion and shall only parse the expansion as a command substitution if it determines that it cannot parse the expansion as an arithmetic expansion. The shell need not evaluate nested expansions when performing this determination. If it encounters the end of input without already having determined that it cannot parse the expansion as an arithmetic expansion, the shell shall treat the expansion as an incomplete arithmetic expansion and report a syntax error. A conforming application shall ensure that it separates the `"$("` and `'('` into two tokens (that is, separate them with white space) in a command substitution that starts with a subshell. For example, a command substitution containing a single subshell could be written as:

```
$( (commands) )
```

2.6.4 Arithmetic Expansion

Arithmetic expansion provides a mechanism for evaluating an arithmetic expression and substituting its value. The format for arithmetic expansion shall be as follows:

```
$((expression))
```

The expression shall be treated as if it were in double-quotes, except that a double-quote inside the expression is not treated specially. The shell shall expand all tokens in the expression for parameter expansion, command substitution, and quote removal.

Next, the shell shall treat this as an arithmetic expression and substitute the value of the expression. The arithmetic expression shall be processed according to the rules given in Section 1.1.2.1, with the following exceptions:

- Only signed long integer arithmetic is required.
- Only the decimal-constant, octal-constant, and hexadecimal-constant constants specified in the ISO C standard, Section 6.4.4.1 are required to be recognized as constants.
- The *sizeof*() operator and the prefix and postfix `"++"` and `"--"` operators are not required.
- Selection, iteration, and jump statements are not supported.

All changes to variables in an arithmetic expression shall be in effect after the arithmetic expansion, as in the parameter expansion `"${x=value}"`.

If the shell variable *x* contains a value that forms a valid integer constant, optionally including a leading <plus-sign> or <hyphen-minus>, then the arithmetic expansions `"$((x))"` and `"$(($x))"` shall return the same value.

As an extension, the shell may recognize arithmetic expressions beyond those listed. The shell may use a signed integer type with a rank larger than the rank of **signed long**. The shell may use a real-floating type instead of **signed long** as long as it does not affect the results in cases where there is no overflow. If the expression is invalid, or the contents of a shell variable used in the expression are not recognized by the shell, the expansion fails and the shell shall write a diagnostic message to standard error indicating the failure.

Examples

A simple example using arithmetic expansion:

```
# repeat a command 100 times
x=100
while [ $x -gt 0 ]
do
    command
    x=$(($x-1))
done
```

2.6.5 Field Splitting

After parameter expansion (Section 2.6.2), command substitution (Section 2.6.3), and arithmetic expansion (Section 2.6.4), if the shell variable *IFS* (see Section 2.5.3) is set and its value is not empty, or if *IFS* is unset, the shell shall scan each field containing results of expansions and substitutions that did not occur in double-quotes for field splitting; zero, one or multiple fields can result.

For the remainder of this section, any reference to the results of an expansion, or results of expansions, shall be interpreted to mean the results from one or more unquoted variable or arithmetic expansions, or unquoted command substitutions.

If the *IFS* variable is set and has an empty string as its value, no field splitting shall occur. However, if an input field which contained the results of an expansion is entirely empty, it shall be removed. Note that this occurs before quote removal; any input field that contains any quoting characters can never be empty at this point. After the removal of any such fields from the input, the possibly modified input field list shall become the output.

Each input field shall be considered in sequence, first to last, with the results of the algorithm described in this section causing output fields to be generated, which shall remain in the same order as the input fields from which they originated.

Fields which contain no results from expansions shall not be affected by field splitting, and shall remain unaltered, simply moving from the list of input fields to be next in the list of output fields.

In the remainder of this description, it is assumed that there is present in the field at least one expansion result; this assumption will not be restated. Field splitting only ever alters those parts of the field.

For the purposes of this section, the term ``*IFS* white space'' is used to mean any of the white-space bytes (see XBD Section 3.413 (on page 92), Section 3.414 (on page 92), and Section 3.415, on page 92) <space>, <tab>, or <newline> from the portable character set (see XBD Section 6.1, on page 117) which are present in the value of the *IFS* variable, and perhaps other white-space characters. It is implementation-defined whether other white-space characters which appear in

the value of *IFS* are also considered as ``*IFS* white space''. The three characters above specified as *IFS* white-space bytes are always *IFS* white space, when they occur in the value of *IFS*, regardless of whether they are white-space characters in any relevant locale. For other locale-specific white-space characters allowed by the implementation it is unspecified whether the character is considered as *IFS* white space if it is white space at the time it is assigned to the *IFS* variable, or if it is white space at the time field splitting occurs. (The locale might have changed between those events.)

If the *IFS* variable is unset, then for the purposes of this section, but without altering the value of the variable, its value shall be considered to contain the three single-byte characters <space>, <tab>, and <newline> from the portable character set, all of which are *IFS* white-space characters.

The shell shall use the byte sequences that form the characters in the value of the *IFS* variable as delimiters. Each of the characters <space>, <tab>, and <newline> which appears in the value of *IFS* shall be a single-byte delimiter. The shell shall use these delimiters as field terminators to split the results of expansions, along with other adjacent bytes, into separate fields, as described below. Note that these delimiters terminate a field; they do not, of themselves, cause a new field to start—subsequent bytes that are not from the results of an expansion, or that do not form *IFS* white-space characters are required for a new field to begin.

Note that the shell processes arbitrary bytes from the input fields; there is no requirement that those bytes form valid characters.

If the results of the algorithm are that no fields are delimited; that is, if the input field is wholly empty or consists entirely of *IFS* white space, the result shall be zero fields (rather than an empty field).

For the purposes of this section, when a field is said to be delimited, then the candidate field, as generated below shall become an output field. When the algorithm transforms a candidate into an output field it shall be appended to the current list of output fields.

Each field containing the results from an expansion shall be processed in order, intermixed with fields not containing the results of expansions, processed as described above, as if by using the following algorithm, examining bytes in the input field, from beginning to end:

- Begin with an empty candidate field and the input as specified above.
- When instructed to start the next iteration of the loop, this is the start of the loop. While the input (as modified by earlier iterations of this loop) is not empty:
  - — Consider the leading remaining byte or byte sequence of the input. No such byte sequence shall contain data such that some bytes in the sequence resulted from an expansion, and others did not, nor which contains bytes resulting from the results of more than one expansion. If the byte or sequence of bytes is:
    1. A byte (or sequence of bytes) in the input which did not result from an expansion:

       Append this byte (or sequence) to the candidate, and remove it from the input. Start the next iteration of the loop.

    2. A byte sequence in the input which resulted from an expansion and which does not form a character in *IFS*:

       Append the first byte of the sequence to the candidate, and remove that byte from the input. Start the next iteration of the loop.

3. A byte sequence in the input which resulted from an expansion and which forms an *IFS* white space character:

   Remove that byte sequence from the input, consider the new leading input byte sequence, and repeat this step.

4. A byte sequence in the input which resulted from an expansion and which forms an *IFS* character that is not *IFS* white space:

   Remove that byte sequence from the input, but note it was observed.

At this point, if the candidate is not empty, or if a sequence of bytes representing an *IFS* character that is not *IFS* white space was seen at step 4, then a field is said to have been delimited, and the candidate shall become an output field.

— Empty (clear) the candidate, and start the next iteration of the loop.

- Once the input is empty, the candidate shall become an output field if and only if it is not empty.

The ordered list of output fields so produced, which might be empty, shall replace the list of input fields.

2.6.6 Pathname Expansion

After field splitting, if *set* **–f** is not in effect, each field in the resulting command line shall be expanded using the algorithm described in Section 2.14, qualified by the rules in Section 2.14.3.

2.6.7 Quote Removal

The quote character sequence <dollar-sign> single-quote and the single-character quote characters (<backslash>, single-quote, and double-quote) that were present in the original word shall be removed unless they have themselves been quoted. Note that the single-quote character that terminates a <dollar-sign> single-quote sequence is itself a single-character quote character.

Note: After quote removal the shell still remembers which characters were quoted. This is necessary for purposes such as matching patterns in a **case** conditional construct (see Section 2.9.4.3 and Section 2.14).

2.7 Redirection

Redirection is used to open and close files for the current shell execution environment (see Section 2.13) or for any command. Redirection operators can be used with numbers representing file descriptors (see XBD Section 3.141) as described below.

The overall format used for redirection is:

```
[n]redir-op word
```

The number *n* is an optional one or more digit decimal number designating the file descriptor number; the application shall ensure it is delimited from any preceding text and immediately precedes the redirection operator *redir-op* (with no intervening <blank> characters allowed). If *n* is quoted, the number shall not be recognized as part of the redirection expression. For example:

```
echo \2>a
```

writes the character 2 into file **a**. If any part of *redir-op* is quoted, no redirection expression is

recognized. For example:

```
echo 2\>a
```

writes the characters 2>*a* to standard output. The optional number, redirection operator, and *word* shall not appear in the arguments provided to the command to be executed (if any).

The shell may support an additional format used for redirection:

```
{location}redir-op word
```

where *location* is non-empty and indicates a location where an integer value can be stored, such as the name of a shell variable. If this format is supported its behavior is implementation-defined.

The largest file descriptor number supported in shell redirections is implementation-defined; however, all implementations shall support at least 0 to 9, inclusive, for use by the application.

If the redirection operator is `"<<"` or `"<<-"`, the word that follows the redirection operator shall be subjected to quote removal; it is unspecified whether any of the other expansions occur. For the other redirection operators, the word that follows the redirection operator shall be subjected to tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote removal. Pathname expansion shall not be performed on the word by a non-interactive shell; an interactive shell may perform it, but if the expansion would result in more than one word it is unspecified whether the redirection proceeds without pathname expansion being performed or the redirection fails.

Note: A future version of this standard may require that the redirection fails in this case.

If more than one redirection operator is specified with a command, the order of evaluation is from beginning to end.

A failure to open or create a file shall cause a redirection to fail.

2.7.1 Redirecting Input

Input redirection shall cause the file whose name results from the expansion of *word* to be opened for reading on the designated file descriptor, or standard input if the file descriptor is not specified.

The general format for redirecting input is:

```
[n]<word
```

where the optional *n* represents the file descriptor number. If the number is omitted, the redirection shall refer to standard input (file descriptor 0).

2.7.2 Redirecting Output

The two general formats for redirecting output are:

```
[n]>word
[n]>|word
```

where the optional *n* represents the file descriptor number. If the number is omitted, the redirection shall refer to standard output (file descriptor 1).

Output redirection using the `'>'` format shall fail if the *noclobber* option is set (see the description of *set* **–C**) and the file named by the expansion of *word* exists and is either a regular file or a symbolic link that resolves to a regular file; it may also fail if the file is a symbolic link

that does not resolve to an existing file. The check for existence, file creation, and open operations shall be performed atomically as is done by the *open*( ) function as defined in System Interfaces volume of POSIX.1-2024 when the O_CREAT and O_EXCL flags are set, except that if the file exists and is a symbolic link, the open operation need not fail with [EEXIST] unless the symbolic link resolves to an existing regular file. Performing these operations atomically ensures that the creation of lock files and unique (often temporary) files is reliable, with important caveats detailed in Section C.2.7.2 (on page 3891). The check for the type of the file need not be performed atomically with the check for existence, file creation, and open operations. If not, there is a potential race condition that may result in a misleading shell diagnostic message when redirection fails. See XRAT Section C.2.7.2 (on page 3891) for more details.

In all other cases (*noclobber* not set, redirection using `'>'` does not fail for the reasons stated above, or redirection using the `">|"` format), output redirection shall cause the file whose name results from the expansion of *word* to be opened for output on the designated file descriptor, or standard output if none is specified. If the file does not exist, it shall be created as an empty file; otherwise, it shall be opened as if the *open*( ) function was called with the O_TRUNC flag set.

2.7.3 Appending Redirected Output

Appended output redirection shall cause the file whose name results from the expansion of word to be opened for output on the designated file descriptor. The file shall be opened as if the *open*( ) function as defined in the System Interfaces volume of POSIX.1-2024 was called with the O_APPEND flag set. If the file does not exist, it shall be created.

The general format for appending redirected output is as follows:

`[`*n*`]>>`*word*

where the optional *n* represents the file descriptor number. If the number is omitted, the redirection refers to standard output (file descriptor 1).

2.7.4 Here-Document

The redirection operators `"<<"` and `"<<-"` both allow redirection of subsequent lines read by the shell to the input of a command. The redirected lines are known as a ``here-document''.

The here-document shall be treated as a single word that begins after the next **NEWLINE** token and continues until there is a line containing only the delimiter and a <newline>, with no <blank> characters in between. Then the next here-document starts, if there is one. For the purposes of locating this terminating line, the end of a *command_string* operand (see *sh*) shall be treated as a <newline> character, and the end of the *commands* string in `$(`*commands*`)` and `` `*commands*` `` may be treated as a <newline>. If the end of input is reached without finding the terminating line, the shell should, but need not, treat this as a redirection error. The format is as follows:

```
[n]<<word
    here-document
delimiter
```

where the optional *n* represents the file descriptor number. If the number is omitted, the here-document refers to standard input (file descriptor 0). It is unspecified whether the file descriptor is opened as a regular file or some other type of file. Portable applications cannot rely on the file descriptor being seekable (see XSH *lseek*( )).

If any part of *word* is quoted, not counting double-quotes outside a command substitution if the here-document is inside one, the delimiter shall be formed by performing quote removal on *word*, and the here-document lines shall not be expanded. Otherwise:

- The delimiter shall be the *word* itself.
- The removal of <backslash><newline> for line continuation (see Section 2.2.1) shall be performed during the search for the trailing delimiter. (As a consequence, the trailing delimiter is not recognized immediately after a <newline> that was removed by line continuation.) It is unspecified whether the line containing the trailing delimiter is itself subject to this line continuation.
- All lines of the here-document shall be expanded, when the redirection operator is evaluated but after the trailing delimiter for the here-document has been located, for parameter expansion, command substitution, and arithmetic expansion. If the redirection operator is never evaluated (because the command it is part of is not executed), the here-document shall be read without performing any expansions.
- Any <backslash> characters in the input shall behave as the <backslash> inside double-quotes (see Section 2.2.3). However, the double-quote character (`'"'`) shall not be treated specially within a here-document, except when the double-quote appears within `"$()"`, `"``"`, or `"${}"`.

If the redirection operator is `"<<-"`, all leading <tab> characters shall be stripped from input lines after <backslash><newline> line continuation (when it applies) has been performed, and from the line containing the trailing delimiter. Stripping of leading <tab> characters shall occur as the here-document is read from the shell input (and consequently does not affect any <tab> characters that result from expansions).

If more than one `"<<"` or `"<<-"` operator is specified on a line, the here-document associated with the first operator shall be supplied first by the application and shall be read first by the shell.

When a here-document is read from a terminal device and the shell is interactive, it shall write the contents of the variable *PS2*, processed as described in Section 2.5.3, to standard error before reading each line of input until the delimiter has been recognized.

Examples

An example of a here-document follows:

```
cat <<eof1; cat <<eof2
Hi,
eof1
Helene.
eof2
```


2.7.5 Duplicating an Input File Descriptor

The redirection operator:

`[n]<&word`

shall duplicate one input file descriptor from another, or shall close one. If *word* evaluates to one or more digits, the file descriptor denoted by *n*, or standard input if *n* is not specified, shall be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent an already open file descriptor, a redirection error shall result (see Section 2.8.1); if the file descriptor denoted by *word* represents an open file descriptor that is not open for input, a redirection error may result. If *word* evaluates to `'-'`, file descriptor *n*, or standard input if *n* is not specified, shall be closed. Attempts to close a file descriptor that is not open shall not constitute an error. If *word* evaluates to something else, the behavior is unspecified.

2.7.6 Duplicating an Output File Descriptor

The redirection operator:

`[n]>&word`

shall duplicate one output file descriptor from another, or shall close one. If *word* evaluates to one or more digits, the file descriptor denoted by *n*, or standard output if *n* is not specified, shall be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent an already open file descriptor, a redirection error shall result (see Section 2.8.1); if the file descriptor denoted by *word* represents an open file descriptor that is not open for output, a redirection error may result. If *word* evaluates to `'-'`, file descriptor *n*, or standard output if *n* is not specified, is closed. Attempts to close a file descriptor that is not open shall not constitute an error. If *word* evaluates to something else, the behavior is unspecified.

2.7.7 Open File Descriptors for Reading and Writing

The redirection operator:

`[n]<>word`

shall cause the file whose name is the expansion of *word* to be opened for both reading and writing on the file descriptor denoted by *n*, or standard input if *n* is not specified. If the file does not exist, it shall be created.

2.8 Exit Status and Errors

2.8.1 Consequences of Shell Errors

Certain errors shall cause the shell to write a diagnostic message to standard error and exit as shown in the following table:

Error	Non-Interactive Shell	Interactive Shell	Shell Diagnostic Message Required
Shell language syntax error	shall exit	shall not exit	yes
Special built-in utility error	shall exit[1]	shall not exit	no[2]
Other utility (not a special built-in) error	shall not exit	shall not exit	no[3]
Redirection error with special built-in utilities	shall exit	shall not exit	yes
Redirection error with compound commands	shall not exit	shall not exit	yes
Redirection error with function execution	shall not exit	shall not exit	yes
Redirection error with other utilities (not special built-ins)	shall not exit	shall not exit	yes
Variable assignment error	shall exit	shall not exit	yes
Expansion error	shall exit	shall not exit	yes
Command not found	may exit	shall not exit	yes
Unrecoverable read error when reading commands	shall exit[4]	shall exit[4]	yes

Notes:

1. The shell shall exit only if the special built-in utility is executed directly. If it is executed via the *command* utility, the shell shall not exit.
2. Although special built-ins are part of the shell, a diagnostic message written by a special built-in is not considered to be a shell diagnostic message, and can be redirected like any other utility.
3. The shell is not required to write a diagnostic message, but the utility itself shall write a diagnostic message if required to do so.
4. If an unrecoverable read error occurs when reading commands, other than from the *file* operand of the *dot* special built-in, the shell shall execute no further commands (including any already successfully read but not yet executed) other than any specified in a previously defined EXIT *trap* action. An unrecoverable read error while reading from the *file* operand of the *dot* special built-in shall be treated as a special built-in utility error.

An expansion error is one that occurs when the shell expansions defined in Section 2.6 are carried out (for example, `"${x!y}"`, because `'!'` is not a valid operator); an implementation may treat these as syntax errors if it is able to detect them during tokenization, rather than during expansion.

If any of the errors shown as ``shall exit'' or ``may exit'' occur in a subshell environment, the shell shall (respectively, may) exit from the subshell environment with a non-zero status and continue in the environment from which that subshell environment was invoked.

In all of the cases shown in the table where an interactive shell is required not to exit and a non-interactive shell is required to exit, an interactive shell shall not perform any further processing of the command in which the error occurred.

2.8.2 Exit Status for Commands

Each command has an exit status that can influence the behavior of other shell commands. The exit status of commands that are not utilities is documented in this section. The exit status of the standard utilities is documented in their respective sections.

The exit status of a command shall be determined as follows:

- If the command is not found, the exit status shall be 127.
- Otherwise, if the command name is found, but it is not an executable utility, the exit status shall be 126.
- Otherwise, if the command terminated due to the receipt of a signal, the shell shall assign it an exit status greater than 128. The exit status shall identify, in an implementation-defined manner, which signal terminated the command. Note that shell implementations are permitted to assign an exit status greater than 255 if a command terminates due to a signal.
- Otherwise, the exit status shall be the value obtained by the equivalent of the WEXITSTATUS macro applied to the status obtained by the *wait*( ) function (as defined in the System Interfaces volume of POSIX.1-2024). Note that for C programs, this value is equal to the result of performing a modulo 256 operation on the value passed to *_Exit*( ), *_exit*( ), or *exit*( ) or returned from *main*( ).

2.9 Shell Commands

This section describes the basic structure of shell commands. The following command descriptions each describe a format of the command that is only used to aid the reader in recognizing the command type, and does not formally represent the syntax. In particular, the representations include spacing between tokens in some places where <blank>s would not be necessary (when one of the tokens is an operator). Each description discusses the semantics of the command; for a formal definition of the command language, consult Section 2.10.

A *command* is one of the following:

- Simple command (see Section 2.9.1)
- Pipeline (see Section 2.9.2)
- List compound-list (see Section 2.9.3)
- Compound command (see Section 2.9.4)
- Function definition (see Section 2.9.5)

Unless otherwise stated, the exit status of a command shall be that of the last simple command executed by the command. There shall be no limit on the size of any shell command other than that imposed by the underlying system (memory constraints, {ARG_MAX}, and so on).

2.9.1 Simple Commands

A ``simple command'' is a sequence of optional variable assignments and redirections, in any sequence, optionally followed by words and redirections.

2.9.1.1 *Order of Processing*

When a given simple command is required to be executed (that is, when any conditional construct such as an AND-OR list or a **case** statement has not bypassed the simple command), the following expansions, assignments, and redirections shall all be performed from the beginning of the command text to the end:

1. The words that are recognized as variable assignments or redirections according to Section 2.10.2 are saved for processing in steps 3 and 4.
2. The first word (if any) that is not a variable assignment or redirection shall be expanded. If any fields remain following its expansion, the first field shall be considered the command name. If no fields remain, the next word (if any) shall be expanded, and so on, until a command name is found or no words remain. If there is a command name and it is recognized as a declaration utility, then any remaining words after the word that expanded to produce the command name, that would be recognized as a variable assignment in isolation, shall be expanded as a variable assignment (tilde expansion after the first <equals-sign> and after any unquoted <colon>, parameter expansion, command substitution, arithmetic expansion, and quote removal, but no field splitting or pathname expansion); while remaining words that would not be a variable assignment in isolation shall be subject to regular expansion (tilde expansion for only a leading <tilde>, parameter expansion, command substitution, arithmetic expansion, field splitting, pathname expansion, and quote removal). For all other command names, words after the word that produced the command name shall be subject only to regular expansion. All fields resulting from the expansion of the word that produced the command name and the subsequent words, except for the field containing the command name, shall be the arguments for the command.
3. Redirections shall be performed as described in Section 2.7.
4. Each variable assignment shall be expanded for tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote removal prior to assigning the value.

In the preceding list, the order of steps 3 and 4 may be reversed if no command name results from step 2 or if the command name matches the name of a special built-in utility; see Section 2.15.

When determining whether a command name is a declaration utility, an implementation may use only lexical analysis. It is unspecified whether assignment context will be used if the command name would only become recognized as a declaration utility after word expansions.

2.9.1.2 *Variable Assignments*

Variable assignments shall be performed as follows:

- If no command name results, variable assignments shall affect the current execution environment.
- If the command name is not a special built-in utility or function, the variable assignments shall be exported for the execution environment of the command and shall not affect the current execution environment except as a side-effect of the expansions performed in step

4. In this case it is unspecified:

   — Whether or not the assignments are visible for subsequent expansions in step 4

   — Whether variable assignments made as side-effects of these expansions are visible for subsequent expansions in step 4, or in the current shell execution environment, or both

- If the command name is a standard utility implemented as a function (see XBD Section 4.25), the effect of variable assignments shall be as if the utility was not implemented as a function.
- If the command name is a special built-in utility, variable assignments shall affect the current execution environment before the utility is executed and remain in effect when the command completes; if an assigned variable is further modified by the utility, the modifications made by the utility shall persist. Unless the *set* **−a** option is on (see *set*), it is unspecified:

   — Whether or not the variables gain the *export* attribute during the execution of the special built-in utility

   — Whether or not *export* attributes gained as a result of the variable assignments persist after the completion of the special built-in utility

- If the command name is a function that is not a standard utility implemented as a function, variable assignments shall affect the current execution environment during the execution of the function. It is unspecified:

   — Whether or not the variable assignments persist after the completion of the function

   — Whether or not the variables gain the *export* attribute during the execution of the function

   — Whether or not *export* attributes gained as a result of the variable assignments persist after the completion of the function (if variable assignments persist after the completion of the function)

If any of the variable assignments attempt to assign a value to a variable for which the *readonly* attribute is set in the current shell environment (regardless of whether the assignment is made in that environment), a variable assignment error shall occur. See Section 2.8.1 for the consequences of these errors.

2.9.1.3 *Commands with no Command Name*

If a simple command has no command name after word expansion (see Section 2.9.1.1), any redirections shall be performed in a subshell environment; it is unspecified whether this subshell environment is the same one as that used for a command substitution within the command. (To affect the current execution environment, see the *exec* special built-in.) If any of the redirections performed in the current shell execution environment fail, the command shall immediately fail with an exit status greater than zero, and the shell shall write an error message indicating the failure. See Section 2.8.1 for the consequences of these failures on interactive and non-interactive shells.

Additionally, if there is no command name but the command contains a command substitution, the command shall complete with the exit status of the command substitution whose exit status was the last to be obtained. Otherwise, the command shall complete with a zero exit status.

81291 *2.9.1.4 Command Search and Execution*

81292 If a simple command has a command name and an optional list of arguments after word
81293 expansion (see Section 2.9.1.1), the following actions shall be performed:

81294 1. If the command name does not contain any <slash> characters, the first successful step in
81295 the following sequence shall occur:

81296 a. If the command name matches the name of a special built-in utility, that special
81297 built-in utility shall be invoked.

81298 b. If the command name matches the name of a utility listed in the following table,
81299 the results are unspecified.

81300 *alloc*	*compcall*	*compvalues*	*history*	*print*
81301 *autoload*	*compctl*	*declare*	*hist*	*pushd*
81302 *bind*	*compdescribe*	*dirs*	*integer*	*readarray*
81303 *bindkey*	*compfiles*	*disable*	*let*	*repeat*
81304 *builtin*	*compgen*	*disown*	*local*	*savehistory*
81305 *bye*	*compgroups*	*dosh*	*login*	*source*
81306 *caller*	*complete*	*echotc*	*logout*	*shopt*
81307 *cap*	*compound*	*echoti*	*map*	*stop*
81308 *chdir*	*compquote*	*enum*	*mapfile*	*suspend*
81309 *clone*	*comptags*	*float*	*nameref*	*typeset*
81310 *comparguments*	*comptry*	*help*	*popd*	*whence*

81311 c. If the command name matches the name of a function known to this shell, the
81312 function shall be invoked as described in Section 2.9.5. If the implementation has
81313 provided a standard utility in the form of a function, and that function definition
81314 still exists (i.e. has not been removed using *unset* **–f** or replaced via another
81315 function definition with the same name), it shall not be recognized at this point. It
81316 shall be invoked in conjunction with the path search in step 1e.

81317 d. If the command name matches the name of an intrinsic utility (see Section 1.7, on
81318 page 2470), that utility shall be invoked.

81319 e. Otherwise, the command shall be searched for using the *PATH* environment
81320 variable as described in XBD Chapter 8:

81321 i. If the search is successful:

81322 a. If the system has implemented the utility as a built-in or as a shell
81323 function, and the built-in or function is associated with the directory
81324 that was most recently tested during the successful *PATH* search, that
81325 built-in or function shall be invoked.

81326 b. Otherwise, the shell shall execute a non-built-in utility as described
81327 in Section 2.9.1.6.

81328 Once a utility has been searched for and found (either as a result of this
81329 specific search or as part of an unspecified shell start-up activity), an
81330 implementation may remember its location and need not search for the
81331 utility again unless the *PATH* variable has been the subject of an assignment.
81332 If the remembered location fails for a subsequent invocation, the shell shall
81333 repeat the search to find the new location for the utility, if any.

81334 ii. If the search is unsuccessful, the command shall fail with an exit status of
81335 127 and the shell shall write an error message.

2. If the command name contains at least one <slash>, the shell shall execute a non-built-in utility as described in Section 2.9.1.6.

2.9.1.5 *Standard File Descriptors*

If the utility would be executed with file descriptor 0, 1, or 2 closed, implementations may execute the utility with the file descriptor open to an unspecified file. If a standard utility or a conforming application is executed with file descriptor 0 not open for reading or with file descriptor 1 or 2 not open for writing, the environment in which the utility or application is executed shall be deemed non-conforming, and consequently the utility or application might not behave as described in this standard.

2.9.1.6 *Non-built-in Utility Execution*

When the shell executes a non-built-in utility, if the execution is not being made via the *exec* special built-in utility, the shell shall execute the utility in a separate utility environment (see Section 2.13).

If the execution is being made via the *exec* special built-in utility, the shell shall not create a separate utility environment for this execution; the new process image shall replace the current shell execution environment. If the current shell environment is a subshell environment, the new process image shall replace the subshell environment and the shell shall continue in the environment from which that subshell environment was invoked.

In either case, execution of the utility in the specified environment shall be performed as follows:

1. If the command name does not contain any <slash> characters, the command name shall be searched for using the *PATH* environment variable as described in XBD Chapter 8 (on page 167):

    a. If the search is successful, the shell shall execute the utility with actions equivalent to calling the *execl*( ) function as defined in the System Interfaces volume of POSIX.1-2024 with the *path* argument set to the pathname resulting from the search, *arg0* set to the command name, and the remaining *execl*( ) arguments set to the command arguments (if any) and the null terminator.

    If the *execl*( ) function fails due to an error equivalent to the [ENOEXEC] error defined in the System Interfaces volume of POSIX.1-2024, the shell shall execute a command equivalent to having a shell invoked with the pathname resulting from the search as its first operand, with any remaining arguments passed to the new shell, except that the value of `"$0"` in the new shell may be set to the command name. The shell may apply a heuristic check to determine if the file to be executed could be a script and may bypass this command execution if it determines that the file cannot be a script. In this case, it shall write an error message, and the command shall fail with an exit status of 126.

    Note: A common heuristic for rejecting files that cannot be a script is locating a NUL byte prior to a <newline> byte within a fixed-length prefix of the file. Since *sh* is required to accept input files with unlimited line lengths, the heuristic check cannot be based on line length.

    It is unspecified whether environment variables that were passed to the shell when it was invoked, but were not used to initialize shell variables (see Section 2.5.3) because they had invalid names, are included in the environment passed to *execl*( ) and (if *execl*( ) fails as described above) to the new shell.

b. If the search is unsuccessful, the command shall fail with an exit status of 127 and the shell shall write an error message.

2. If the command name contains at least one <slash>:

a. If the named utility exists, the shell shall execute the utility with actions equivalent to calling the *execl*( ) function defined in the System Interfaces volume of POSIX.1-2024 with the *path* and *arg0* arguments set to the command name, and the remaining *execl*( ) arguments set to the command arguments (if any) and the null terminator.

If the *execl*( ) function fails due to an error equivalent to the [ENOEXEC] error, the shell shall execute a command equivalent to having a shell invoked with the command name as its first operand, with any remaining arguments passed to the new shell. The shell may apply a heuristic check to determine if the file to be executed could be a script and may bypass this command execution if it determines that the file cannot be a script. In this case, it shall write an error message, and the command shall fail with an exit status of 126.

Note: A common heuristic for rejecting files that cannot be a script is locating a NUL byte prior to a <newline> byte within a fixed-length prefix of the file. Since *sh* is required to accept input files with unlimited line lengths, the heuristic check cannot be based on line length.

It is unspecified whether environment variables that were passed to the shell when it was invoked, but were not used to initialize shell variables (see Section 2.5.3) because they had invalid names, are included in the environment passed to *execl*( ) and (if *execl*( ) fails as described above) to the new shell.

b. If the named utility does not exist, the command shall fail with an exit status of 127 and the shell shall write an error message.

2.9.2 Pipelines

A *pipeline* is a sequence of one or more commands separated by the control operator '|'. For each command but the last, the shell shall connect the standard output of the command to the standard input of the next command as if by creating a pipe and passing the write end of the pipe as the standard output of the command and the read end of the pipe as the standard input of the next command.

The format for a pipeline is:

[!] *command1* **[** | *command2* **...]**

If the pipeline begins with the reserved word **!** and *command1* is a subshell command, the application shall ensure that the **(** operator at the beginning of *command1* is separated from the **!** by one or more <blank> characters. The behavior of the reserved word **!** immediately followed by the **(** operator is unspecified.

The standard output of *command1* shall be connected to the standard input of *command2*. The standard input, standard output, or both of a command shall be considered to be assigned by the pipeline before any redirection specified by redirection operators that are part of the command (see Section 2.7).

If the pipeline is not in the background (see Section 2.9.3.1 and Section 2.11), the shell shall wait for the last command specified in the pipeline to complete, and may also wait for all commands to complete.

81424 **Exit Status**

81425 The exit status of a pipeline shall depend on whether or not the *pipefail* option (see *set*) is enabled
81426 and whether or not the pipeline begins with the **!** reserved word, as described in the following
81427 table. The *pipefail* option determines which command in the pipeline the exit status is derived
81428 from; the **!** reserved word causes the exit status to be the logical NOT of the exit status of that
81429 command. The shell shall use the *pipefail* setting at the time it begins execution of the pipeline,
81430 not the setting at the time it sets the exit status of the pipeline. (For example, in `command1 |`
81431 `set -o pipefail` the exit status of `command1` has no effect on the exit status of the pipeline,
81432 even if the shell executes `set -o pipefail` in the current shell environment.)

81433

pipefail Enabled	Begins with !	Exit Status
81434 81435 no	no	The exit status of the last (rightmost) command specified in the pipeline.
81436 81437 no	yes	Zero, if the last (rightmost) command in the pipeline returned a non-zero exit status; otherwise, 1.
81438 81439 81440 81441 yes	no	Zero, if all commands in the pipeline returned an exit status of 0; otherwise, the exit status of the last (rightmost) command specified in the pipeline that returned a non-zero exit status.
81442 81443 yes	yes	Zero, if any command in the pipeline returned a non-zero exit status; otherwise, 1.

81444 ### 2.9.3 Lists

81445 An *AND-OR list* is a sequence of one or more pipelines separated by the operators `"&&"` and
81446 `"||"`.

81447 A *list* is a sequence of one or more AND-OR lists separated by the operators `';'` and `'&'`.

81448 The operators `"&&"` and `"||"` shall have equal precedence and shall be evaluated with left
81449 associativity. For example, both of the following commands write solely **bar** to standard output:

```
81450 false && echo foo || echo bar
81451 true || echo foo && echo bar
```

81452 A `';'` separator or a `';'` or <newline> terminator shall cause the preceding AND-OR list to be
81453 executed sequentially; an `'&'` separator or terminator shall cause asynchronous execution of the
81454 preceding AND-OR list.

81455 The term ``compound-list'' is derived from the grammar in Section 2.10; it is equivalent to a
81456 sequence of *lists*, separated by <newline> characters, that can be preceded or followed by an
81457 arbitrary number of <newline> characters.

81458 **Examples**

81459 The following is an example that illustrates <newline> characters in compound-lists:

```
81460 while
81461     # a couple of <newline>s
81462     # a list
81463     date && who || ls; cat file
81464     # a couple of <newline>s
81465     # another list
```



```
        wc file > output & true
do
        # 2 lists
        ls
        cat file
done
```

2.9.3.1 Asynchronous AND-OR Lists

If an AND-OR list is terminated by the control operator <ampersand> (`'&'`), the shell shall execute the AND-OR list asynchronously in a subshell environment. This subshell shall execute in the background; that is, the shell shall not wait for the subshell to terminate before executing the next command (if any); if there are no further commands to execute, the shell shall not wait for the subshell to terminate before exiting.

If job control is enabled (see *set*, **−m**), the AND-OR list shall become a job-control background job and a job number shall be assigned to it. If job control is disabled, the AND-OR list may become a non-job-control background job, in which case a job number shall be assigned to it; if no job number is assigned it shall become a background command but not a background job.

A job-control background job can be controlled as described in Section 2.11.

The process ID associated with the asynchronous AND-OR list shall become known in the current shell execution environment; see Section 2.13. This process ID shall remain known until any one of the following occurs (and, unless otherwise specified, may continue to remain known after it occurs).

- The process terminates and the application waits for the process ID or the corresponding job ID (see *wait*).
- If the asynchronous AND-OR list did not become a background job: another asynchronous AND-OR list is invoked before `"$!"` (corresponding to the previous asynchronous AND-OR list) is expanded in the current shell execution environment.
- If the asynchronous AND-OR list became a background job: the *jobs* utility reports the termination status of that job.
- If the shell is interactive and the asynchronous AND-OR list became a background job: a message indicating completion of the corresponding job is written to standard error. If *set* **−b** is enabled, it is unspecified whether the process ID is removed from the list of known process IDs when the message is written or immediately prior to when the shell writes the next prompt for input.

The implementation need not retain more than the {CHILD_MAX} most recent entries in its list of known process IDs in the current shell execution environment.

If, and only if, job control is disabled, the standard input for the subshell in which an asynchronous AND-OR list is executed shall initially be assigned to an open file description that behaves as if **/dev/null** had been opened for reading only. This initial assignment shall be overridden by any explicit redirection of standard input within the AND-OR list.

If the shell is interactive and the asynchronous AND-OR list became a background job, the job number and the process ID associated with the job shall be written to standard error using the format:

`"[%d] %d\n", <job-number>, <process-id>`

If the shell is interactive and the asynchronous AND-OR list did not become a background job, the process ID associated with the asynchronous AND-OR list shall be written to standard error in an unspecified format.

Exit Status

The exit status of an asynchronous AND-OR list shall be zero.

The exit status of the subshell in which the AND-OR list is asynchronously executed can be obtained using the *wait* utility.

2.9.3.2 Sequential AND-OR Lists

AND-OR lists that are separated by a <semicolon> (`';'`) shall be executed sequentially. The format for executing AND-OR lists sequentially shall be:

aolist1 **[;** *aolist2***]** **...**

Each AND-OR list shall be expanded and executed in the order specified.

If job control is enabled, the AND-OR lists shall form all or part of a foreground job that can be controlled as described in Section 2.11.

Exit Status

The exit status of a sequential AND-OR list shall be the exit status of the last pipeline in the AND-OR list that is executed.

2.9.3.3 AND Lists

The control operator `"&&"` denotes an AND list. The format shall be:

command1 **[** `&&` *command2***]** **...**

First *command1* shall be executed. If its exit status is zero, *command2* shall be executed, and so on, until a command has a non-zero exit status or there are no more commands left to execute. The commands are expanded only if they are executed.

Exit Status

The exit status of an AND list shall be the exit status of the last command that is executed in the list.

2.9.3.4 OR Lists

The control operator `"||"` denotes an OR List. The format shall be:

command1 **[** `||` *command2***]** **...**

First, *command1* shall be executed. If its exit status is non-zero, *command2* shall be executed, and so on, until a command has a zero exit status or there are no more commands left to execute.

81540 **Exit Status**

81541 The exit status of an OR list shall be the exit status of the last command that is executed in the
81542 list.

81543 ## 2.9.4 Compound Commands

81544 The shell has several programming constructs that are ``compound commands'', which provide
81545 control flow for commands. Each of these compound commands has a reserved word or control
81546 operator at the beginning, and a corresponding terminator reserved word or operator at the end.
81547 In addition, each can be followed by redirections on the same line as the terminator. Each
81548 redirection shall apply to all the commands within the compound command that do not
81549 explicitly override that redirection.

81550 In the descriptions below, the exit status of some compound commands is stated in terms of the
81551 exit status of a *compound-list*. The exit status of a *compound-list* shall be the value that the special
81552 parameter `'?'` (see Section 2.5.2) would have immediately after execution of the *compound-list*.

81553 ### 2.9.4.1 *Grouping Commands*

81554 The format for grouping commands is as follows:

81555 (*compound-list*) Execute *compound-list* in a subshell environment; see Section 2.13.
81556 Variable assignments and built-in commands that affect the environment
81557 shall not remain in effect after the list finishes.

81558 If a character sequence beginning with `"(("` would be parsed by the shell
81559 as an arithmetic expansion if preceded by a `'$'`, shells which implement
81560 an extension whereby `"((`*expression*`))"` is evaluated as an arithmetic
81561 expression may treat the `"(("` as introducing as an arithmetic evaluation
81562 instead of a grouping command. A conforming application shall ensure
81563 that it separates the two leading `'('` characters with white space to
81564 prevent the shell from performing an arithmetic evaluation.

81565 { *compound-list* ; } Execute *compound-list* in the current process environment. The semicolon
81566 shown here is an example of a control operator delimiting the **}** reserved
81567 word. Other delimiters are possible, as shown in Section 2.10; a
81568 <newline> is frequently used.

81569 **Exit Status**

81570 The exit status of a grouping command shall be the exit status of *compound-list*.

81571 ### 2.9.4.2 *The for Loop*

81572 The **for** loop shall execute a sequence of commands for each member in a list of *items*. The **for**
81573 loop requires that the reserved words **do** and **done** be used to delimit the sequence of
81574 commands.

81575 The format for the **for** loop is as follows:

```
81576 for name [ in [word ... ]]
81577 do
81578     compound-list
81579 done
```


First, the list of words following **in** shall be expanded to generate a list of items. Then, the variable *name* shall be set to each item, in turn, and the *compound-list* executed each time. If no items result from the expansion, the *compound-list* shall not be executed. Omitting:

```
in word...
```

shall be equivalent to:

```
in "$@"
```

Exit Status

If there is at least one item in the list of items, the exit status of a **for** command shall be the exit status of the last *compound-list* executed. If there are no items, the exit status shall be zero.

2.9.4.3 *Case Conditional Construct*

The conditional construct **case** shall execute the *compound-list* corresponding to the first *pattern* (see Section 2.14), if any are present, that is matched by the string resulting from the tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote removal of the given word. The reserved word **in** shall denote the beginning of the patterns to be matched. Multiple patterns with the same *compound-list* shall be delimited by the '|' symbol. The control operator ')' terminates a list of patterns corresponding to a given action. The terminated pattern list and the following *compound-list* is called a **case** statement *clause*. Each **case** statement clause, with the possible exception of the last, shall be terminated with either ";;" or ";&". The **case** construct terminates with the reserved word **esac** (**case** reversed).

The format for the **case** construct is as follows:

```
case word in
    [[(] pattern[ | pattern] ... ) compound-list terminator] ...
    [[(] pattern[ | pattern] ... ) compound-list]
esac
```

Where *terminator* is either ";;" or ";&" and is optional for the last *compound-list*.

In order from the beginning to the end of the **case** statement, each *pattern* that labels a *compound-list* shall be subjected to tilde expansion, parameter expansion, command substitution, and arithmetic expansion, and the result of these expansions shall be compared against the expansion of *word*, according to the rules described in Section 2.14 (which also describes the effect of quoting parts of the pattern). After the first match, no more patterns in the **case** statement shall be expanded, and the *compound-list* of the matching clause shall be executed. If the **case** statement clause is terminated by ";;", no further clauses shall be examined. If the **case** statement clause is terminated by ";&", then the *compound-list* (if any) of each subsequent clause shall be executed, in order, until either a clause terminated by ";;" is reached and its *compound-list* (if any) executed or there are no further clauses in the **case** statement. The order of expansion and comparison of multiple *pattern*s that label a *compound-list* statement is unspecified.

81617 **Exit Status**

81618 The exit status of **case** shall be zero if no patterns are matched. Otherwise, the exit status shall be
81619 the exit status of the *compound-list* of the last clause to be executed.

81620 *2.9.4.4 The if Conditional Construct*

81621 The **if** command shall execute a *compound-list* and use its exit status to determine whether to
81622 execute another *compound-list*.

81623 The format for the **if** construct is as follows:

```
81624 if compound-list
81625 then
81626     compound-list
81627 [elif compound-list
81628 then
81629     compound-list] ...
81630 [else
81631     compound-list]
81632 fi
```

81633 The **if** *compound-list* shall be executed; if its exit status is zero, the **then** *compound-list* shall be
81634 executed and the command shall complete. Otherwise, each **elif** *compound-list* shall be executed,
81635 in turn, and if its exit status is zero, the **then** *compound-list* shall be executed and the command
81636 shall complete. Otherwise, the **else** *compound-list* shall be executed.

81637 **Exit Status**

81638 The exit status of the **if** command shall be the exit status of the **then** or **else** *compound-list* that
81639 was executed, or zero, if none was executed.

81640 **Note:** Although the exit status of the **if** or **elif** *compound-list* is ignored when determining the exit
81641 status of the **if** command, it is available through the special parameter `'?'` (see Section 2.5.2)
81642 during execution of the next **then**, **elif**, or **else** *compound-list* (if any is executed) in the normal
81643 way.

81644 *2.9.4.5 The while Loop*

81645 The **while** loop shall continuously execute one *compound-list* as long as another *compound-list* has
81646 a zero exit status.

81647 The format of the **while** loop is as follows:

```
81648 while compound-list-1
81649 do
81650     compound-list-2
81651 done
```

81652 The *compound-list-1* shall be executed, and if it has a non-zero exit status, the **while** command
81653 shall complete. Otherwise, the *compound-list-2* shall be executed, and the process shall repeat.

81654 **Exit Status**

81655 The exit status of the **while** loop shall be the exit status of the last *compound-list-2* executed, or
81656 zero if none was executed.

81657 **Note:** Since the exit status of *compound-list-1* is ignored when determining the exit status of the **while**
81658 command, it is not possible to obtain the status of the command that caused the loop to exit,
81659 other than via the special parameter `'?'` (see Section 2.5.2) during execution of *compound-list-1*,
81660 for example:

81661 `while some_command; st=$?; false; do ...`

81662 The exit status of *compound-list-1* is available through the special parameter `'?'` during
81663 execution of *compound-list-2*, but is known to be zero at that point anyway.

81664 *2.9.4.6 The until Loop*

81665 The **until** loop shall continuously execute one *compound-list* as long as another *compound-list* has
81666 a non-zero exit status.

81667 The format of the **until** loop is as follows:

```
81668 until compound-list-1
81669 do
81670     compound-list-2
81671 done
```

81672 The *compound-list-1* shall be executed, and if it has a zero exit status, the **until** command
81673 completes. Otherwise, the *compound-list-2* shall be executed, and the process repeats.

81674 **Exit Status**

81675 The exit status of the **until** loop shall be the exit status of the last *compound-list-2* executed, or
81676 zero if none was executed.

81677 **Note:** Although the exit status of *compound-list-1* is ignored when determining the exit status of the
81678 **until** command, it is available through the special parameter `'?'` (see Section 2.5.2) during
81679 execution of *compound-list-2* in the normal way.

81680 ## 2.9.5 Function Definition Command

81681 A function is a user-defined name that is used as a simple command to call a compound
81682 command with new positional parameters. A function is defined with a ``function definition
81683 command''.

81684 The format of a function definition command is as follows:

81685 *fname* `( )` *compound-command* `[`*io-redirect* `...]`

81686 The function is named *fname*; the application shall ensure that it is a name (see XBD Section
81687 3.216) and that it is not the name of a special built-in utility. An implementation may allow other
81688 characters in a function name as an extension. The implementation shall maintain separate name
81689 spaces for functions and variables.

81690 The argument *compound-command* represents a compound command, as described in Section
81691 2.9.4.

81692 When the function is declared, none of the expansions in Section 2.6 shall be performed on the
81693 text in *compound-command* or *io-redirect*; all expansions shall be performed as normal each time

the function is called. Similarly, the optional *io-redirect* redirections and any variable assignments within *compound-command* shall be performed during the execution of the function itself, not the function definition. See Section 2.8.1 for the consequences of failures of these operations on interactive and non-interactive shells.

When a function is executed, it shall have the syntax-error properties described for special built-in utilities in the first item in the enumerated list at the beginning of Section 2.15.

The *compound-command* shall be executed whenever the function name is specified as the name of a simple command (see Section 2.9.1.4). The operands to the command temporarily shall become the positional parameters during the execution of the *compound-command*; the special parameter `'#'` also shall be changed to reflect the number of operands. The special parameter 0 shall be unchanged. When the function completes, the values of the positional parameters and the special parameter `'#'` shall be restored to the values they had before the function was executed. If the special built-in *return* (see *return*) is executed in the *compound-command*, the function completes and execution shall resume with the next command after the function call.

Exit Status

The exit status of a function definition shall be zero if the function was declared successfully; otherwise, it shall be greater than zero. The exit status of a function invocation shall be the exit status of the last command executed by the function.

2.10 Shell Grammar

The following grammar defines the Shell Command Language. This formal syntax shall take precedence over the preceding text syntax description.

2.10.1 Shell Grammar Lexical Conventions

The input language to the shell shall be first recognized at the character level. The resulting tokens shall be classified by their immediate context according to the following rules (applied in order). These rules shall be used to determine what a ``token'' is that is subject to parsing at the token level. The rules for token recognition in Section 2.3 shall apply.

1. If the token is an operator, the token identifier for that operator shall result.
2. If the string consists solely of digits and the delimiter character is one of `'<'` or `'>'`, the token identifier **IO_NUMBER** shall result.
3. If the string contains at least three characters, begins with a <left-curly-bracket> (`'{'`) and ends with a <right-curly-bracket> (`'}'`), and the delimiter character is one of `'<'` or `'>'`, the token identifier **IO_LOCATION** may result; if the result is not **IO_LOCATION**, the token identifier **TOKEN** shall result.
4. Otherwise, the token identifier **TOKEN** shall result.

Further distinction on **TOKEN** is context-dependent. It may be that the same **TOKEN** yields **WORD**, a **NAME**, an **ASSIGNMENT_WORD**, or one of the reserved words below, dependent upon the context. Some of the productions in the grammar below are annotated with a rule number from the following list. When a **TOKEN** is seen where one of those annotated productions could be used to reduce the symbol, the applicable rule shall be applied to convert the token identifier type of the **TOKEN** to:

- The token identifier of the recognized reserved word, for rule 1
- A token identifier acceptable at that point in the grammar, for all other rules

The reduction shall then proceed based upon the token identifier type yielded by the rule applied. When more than one rule applies, the highest numbered rule shall apply (which in turn may refer to another rule). (Note that except in rule 7, the presence of an `'='` in the token has no effect.)

The **WORD** tokens shall have the word expansion rules applied to them immediately before the associated command is executed, not at the time the command is parsed.

2.10.2 Shell Grammar Rules

1. [Command Name]

   When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word shall result. Otherwise, the token **WORD** shall be returned. Also, if the parser is in any state where only a reserved word could be the next correct token, proceed as above.

   Note: Because at this point quoting characters (<backslash>, single-quote, <quotation-mark>, and the <dollar-sign> single-quote sequence) are retained in the token, quoted strings cannot be recognized as reserved words. This rule also implies that reserved words are not recognized except in certain positions in the input, such as after a <newline> or <semicolon>; the grammar presumes that if the reserved word is intended, it is properly delimited by the user, and does not attempt to reflect that requirement directly. Also note that line joining is done before tokenization, as described in Section 2.2.1, so escaped <newline> characters are already removed at this point.

   Rule 1 is not directly referenced in the grammar, but is referred to by other rules, or applies globally.

2. [Redirection to or from filename]

   The expansions specified in Section 2.7 shall occur. As specified there, exactly one field can result (or the result is unspecified), and there are additional requirements on pathname expansion.

3. [Redirection from here-document]

   Quote removal shall be applied to the word to determine the delimiter that is used to find the end of the here-document that begins after the next <newline>.

4. [Case statement termination]

   When the **TOKEN** is exactly the reserved word **esac**, the token identifier for **esac** shall result. Otherwise, the token **WORD** shall be returned.

5. [**NAME** in **for**]

   When the **TOKEN** meets the requirements for a name (see XBD Section 3.216), the token identifier **NAME** shall result. Otherwise, the token **WORD** shall be returned.

6. [Third word of **for** and **case**]

   a. [**case** only]

      When the **TOKEN** is exactly the reserved word **in**, the token identifier for **in** shall result. Otherwise, the token **WORD** shall be returned.

b. [**for** only]

When the **TOKEN** is exactly the reserved word **in** or **do**, the token identifier for **in** or **do** shall result, respectively. Otherwise, the token **WORD** shall be returned.

(For a. and b.: As indicated in the grammar, a *linebreak* precedes the tokens **in** and **do**. If <newline> characters are present at the indicated location, it is the token after them that is treated in this fashion.)

7. [Assignment preceding command name]

a. [When the first word]

If the **TOKEN** is exactly a reserved word, the token identifier for that reserved word shall result. Otherwise, 7b shall be applied.

b. [Not the first word]

If the **TOKEN** contains an unquoted (as determined while applying rule 4 from Section 2.3) <equals-sign> character that is not part of an embedded parameter expansion, command substitution, or arithmetic expansion construct (as determined while applying rule 5 from Section 2.3):

— If the **TOKEN** begins with `'='`, then the token **WORD** shall be returned.

— If all the characters in the **TOKEN** preceding the first such <equals-sign> form a valid name (see XBD Section 3.216, on page 63), the token **ASSIGNMENT_WORD** shall be returned.

— Otherwise, it is implementation-defined whether the token **WORD** or **ASSIGNMENT_WORD** is returned, or the **TOKEN** is processed in some other way.

Otherwise, the token **WORD** shall be returned.

If a returned **ASSIGNMENT_WORD** token begins with a valid name, assignment of the value after the first <equals-sign> to the name shall occur as specified in Section 2.9.1. If a returned **ASSIGNMENT_WORD** token does not begin with a valid name, the way in which the token is processed is unspecified.

8. [**NAME** in function]

When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word shall result. Otherwise, when the **TOKEN** meets the requirements for a name, the token identifier **NAME** shall result. Otherwise, rule 7 applies.

9. [Body of function]

Word expansion and assignment shall never occur, even when required by the rules above, when this rule is being parsed. Each **TOKEN** that might either be expanded or have assignment applied to it shall instead be returned as a single **WORD** consisting only of characters that are exactly the token described in Section 2.3.


```
81810 /* ------------------------------------------------------
81811    The grammar symbols
81812    ------------------------------------------------------ */
81813 %token  WORD
81814 %token  ASSIGNMENT_WORD
81815 %token  NAME
81816 %token  NEWLINE
81817 %token  IO_NUMBER
81818 %token  IO_LOCATION
81819 /* The following are the operators (see XBD Section 3.243)
81820    containing more than one character. */
81821 %token  AND_IF    OR_IF    DSEMI    SEMI_AND
81822 /*      '&&'      '||'     ';;'     ';&'   */
81823 %token  DLESS  DGREAT  LESSAND  GREATAND  LESSGREAT  DLESSDASH
81824 /*      '<<'   '>>'    '<&'     '>&'      '<>'       '<<-'   */
81825 %token  CLOBBER
81826 /*      '>|'   */
81827 /* The following are the reserved words. */
81828 %token  If    Then    Else    Elif    Fi    Do    Done
81829 /*      'if'  'then'  'else'  'elif'  'fi'  'do'  'done'   */
81830 %token  Case    Esac    While    Until    For
81831 /*      'case'  'esac'  'while'  'until'  'for'   */
81832 /* These are reserved words, not operator tokens, and are
81833    recognized when reserved words are recognized. */
81834 %token  Lbrace    Rbrace    Bang
81835 /*      '{'       '}'       '!'   */
81836 %token  In
81837 /*      'in'   */
81838 /* ------------------------------------------------------
81839    The Grammar
81840    ------------------------------------------------------ */
81841 %start program
81842 %%
81843 program          : linebreak complete_commands linebreak
81844                  | linebreak
81845                  ;
81846 complete_commands: complete_commands newline_list complete_command
81847                  |                                 complete_command
81848                  ;
81849 complete_command : list separator_op
81850                  | list
81851                  ;
81852 list             : list separator_op and_or
81853                  |                   and_or
81854                  ;
81855 and_or           :                         pipeline
81856                  | and_or AND_IF linebreak pipeline
```



```
81857                       | and_or OR_IF  linebreak pipeline
81858                       ;
81859       pipeline        :      pipe_sequence
81860                       | Bang pipe_sequence
81861                       ;
81862       pipe_sequence   :                             command
81863                       | pipe_sequence '|' linebreak command
81864                       ;
81865       command         : simple_command
81866                       | compound_command
81867                       | compound_command redirect_list
81868                       | function_definition
81869                       ;
81870       compound_command : brace_group
81871                       | subshell
81872                       | for_clause
81873                       | case_clause
81874                       | if_clause
81875                       | while_clause
81876                       | until_clause
81877                       ;
81878       subshell        : '(' compound_list ')'
81879                       ;
81880       compound_list   : linebreak term
81881                       | linebreak term separator
81882                       ;
81883       term            : term separator and_or
81884                       |                and_or
81885                       ;
81886       for_clause      : For name                                      do_group
81887                       | For name                       sequential_sep do_group
81888                       | For name linebreak in          sequential_sep do_group
81889                       | For name linebreak in wordlist sequential_sep do_group
81890                       ;
81891       name            : NAME                     /* Apply rule 5 */
81892                       ;
81893       in              : In                       /* Apply rule 6 */
81894                       ;
81895       wordlist        : wordlist WORD
81896                       |          WORD
81897                       ;
81898       case_clause     : Case WORD linebreak in linebreak case_list    Esac
81899                       | Case WORD linebreak in linebreak case_list_ns Esac
81900                       | Case WORD linebreak in linebreak              Esac
81901                       ;
81902       case_list_ns    : case_list case_item_ns
81903                       |           case_item_ns
81904                       ;
81905       case_list       : case_list case_item
81906                       |           case_item
81907                       ;
81908       case_item_ns    : pattern_list ')' linebreak
81909                       | pattern_list ')' compound_list
```



```
81910                       ;
81911       case_item       : pattern_list ')' linebreak     DSEMI linebreak
81912                       | pattern_list ')' compound_list DSEMI linebreak
81913                       | pattern_list ')' linebreak     SEMI_AND linebreak
81914                       | pattern_list ')' compound_list SEMI_AND linebreak
81915                       ;
81916       pattern_list    :                  WORD    /* Apply rule 4 */
81917                       |              '(' WORD    /* Do not apply rule 4 */
81918                       | pattern_list '|' WORD    /* Do not apply rule 4 */
81919                       ;
81920       if_clause       : If compound_list Then compound_list else_part Fi
81921                       | If compound_list Then compound_list           Fi
81922                       ;
81923       else_part       : Elif compound_list Then compound_list
81924                       | Elif compound_list Then compound_list else_part
81925                       | Else compound_list
81926                       ;
81927       while_clause    : While compound_list do_group
81928                       ;
81929       until_clause    : Until compound_list do_group
81930                       ;
81931       function_definition : fname '(' ')' linebreak function_body
81932                       ;
81933       function_body   : compound_command                /* Apply rule 9 */
81934                       | compound_command redirect_list  /* Apply rule 9 */
81935                       ;
81936       fname           : NAME                            /* Apply rule 8 */
81937                       ;
81938       brace_group     : Lbrace compound_list Rbrace
81939                       ;
81940       do_group        : Do compound_list Done           /* Apply rule 6 */
81941                       ;
81942       simple_command  : cmd_prefix cmd_word cmd_suffix
81943                       | cmd_prefix cmd_word
81944                       | cmd_prefix
81945                       | cmd_name cmd_suffix
81946                       | cmd_name
81947                       ;
81948       cmd_name        : WORD                   /* Apply rule 7a */
81949                       ;
81950       cmd_word        : WORD                   /* Apply rule 7b */
81951                       ;
81952       cmd_prefix      :            io_redirect
81953                       | cmd_prefix io_redirect
81954                       |            ASSIGNMENT_WORD
81955                       | cmd_prefix ASSIGNMENT_WORD
81956                       ;
81957       cmd_suffix      :            io_redirect
81958                       | cmd_suffix io_redirect
81959                       |            WORD
81960                       | cmd_suffix WORD
81961                       ;
81962       redirect_list   :               io_redirect
```



```
                        | redirect_list io_redirect
                        ;
io_redirect             :             io_file
                        | IO_NUMBER   io_file
                        | IO_LOCATION io_file /* Optionally supported */
                        |             io_here
                        | IO_NUMBER   io_here
                        | IO_LOCATION io_here /* Optionally supported */
                        ;
io_file                 : '<'       filename
                        | LESSAND   filename
                        | '>'       filename
                        | GREATAND  filename
                        | DGREAT    filename
                        | LESSGREAT filename
                        | CLOBBER   filename
                        ;
filename                : WORD                      /* Apply rule 2 */
                        ;
io_here                 : DLESS     here_end
                        | DLESSDASH here_end
                        ;
here_end                : WORD                      /* Apply rule 3 */
                        ;
newline_list            :              NEWLINE
                        | newline_list NEWLINE
                        ;
linebreak               : newline_list
                        | /* empty */
                        ;
separator_op            : '&'
                        | ';'
                        ;
separator               : separator_op linebreak
                        | newline_list
                        ;
sequential_sep          : ';' linebreak
                        | newline_list
                        ;
```

2.11 Job Control

Job control is defined (see XBD Section 3.181, on page 57) as a facility that allows users selectively to stop (suspend) the execution of processes and continue (resume) their execution at a later point. It is jointly supplied by the terminal I/O driver and a command interpreter. The shell is one such command interpreter and job control in the shell is enabled by *set* **−m** (which is enabled by default in interactive shells). The remainder of this section describes the job control facility provided by the shell. Requirements relating to background jobs stated in this section only apply to job-control background jobs.

If the shell has a controlling terminal and it is the controlling process for the terminal session, it shall initially set the foreground process group ID associated with the terminal to its own

process group ID. Otherwise, if it has a controlling terminal, it shall initially perform the following steps if interactive and may perform them if non-interactive:

1. If its process group is the foreground process group associated with the terminal, the shell shall set its process group ID to its process ID (if they are not already equal) and set the foreground process group ID associated with the terminal to its process group ID.
2. If its process group is not the foreground process group associated with the terminal (which would result from it being started by a job-control shell as a background job), the shell shall either stop itself by sending itself a SIGTTIN signal or, if interactive, attempt to read from standard input (which generates a SIGTTIN signal if standard input is the controlling terminal). If it is stopped, then when it continues execution (after receiving a SIGCONT signal) it shall repeat these steps.

Subsequently, the shell shall change the foreground process group associated with its controlling terminal when a foreground job is running as noted in the description below.

When job control is enabled, the shell shall create one or more jobs when it executes a list (see Section 2.9.3) that has one of the following forms:

- A single asynchronous AND-OR list
- One or more sequentially executed AND-OR lists followed by at most one asynchronous AND-OR list

For the purposes of job control, a list that includes more than one asynchronous AND-OR list shall be treated as if it were split into multiple separate lists, each ending with an asynchronous AND-OR list.

When a job consisting of a single asynchronous AND-OR list is created, it shall form a *background job* and the associated process ID shall be that of a child process that is made a process group leader, with all other processes (if any) that the shell creates to execute the AND-OR list initially having this process ID as their process group ID.

For a list consisting of one or more sequentially executed AND-OR lists followed by at most one asynchronous AND-OR list, the whole list shall form a single *foreground job* up until the sequentially executed AND-OR lists have all completed execution, at which point the asynchronous AND-OR list (if any) shall form a background job as described above.

For each pipeline in a foreground job, if the pipeline is executed while the list is still a foreground job, the set of processes comprising the pipeline, and any processes descended from it, shall all be in the same process group, unless the shell executes some of the commands in the pipeline in the current shell execution environment and others in a subshell environment; in this case the process group ID of the current shell need not change (or cannot change if it is the session leader), and consequently the process group ID that the other processes all share may differ from the process group ID of the current shell (which means that a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal sent to one of those process groups does not cause the whole pipeline to stop).

A background job that was created on execution of an asynchronous AND-OR list can be brought into the foreground by means of the *fg* utility (if supported); in this case the entire job shall become a single foreground job. If a process that the shell subsequently waits for is part of this foreground job and is stopped by a signal, the entire job shall become a suspended job and the behavior shall be as if the process had been stopped while the job was running in the background.

When a foreground job is created, or a background job is brought into the foreground by the *fg* utility, if the shell has a controlling terminal it shall set the foreground process group ID associated with the terminal as follows:

- If the job was originally created as a background job, the foreground process group ID shall be set to the process ID of the process that the shell made a process group leader when it executed the asynchronous AND-OR list.
- If the job was originally created as a foreground job, the foreground process group ID shall be set as follows when each pipeline in the job is executed:
  - — If the shell is not itself executing, in the current shell execution environment, all of the commands in the pipeline, the foreground process group ID shall be set to the process group ID that is shared by the other processes executing the pipeline (see above).
  - — If all of the commands in the pipeline are being executed by the shell itself in the current shell execution environment, the foreground process group ID shall be set to the process group ID of the shell.

When a foreground job terminates, or becomes a suspended job (see below), if the shell has a controlling terminal it shall set the foreground process group ID associated with the terminal to the process group ID of the shell.

Each background job (whether suspended or not) shall have associated with it a job number and a process ID that is known in the current shell execution environment. When a background job is brought into the foreground by means of the *fg* utility, the associated job number shall be removed from the shell's background jobs list and the associated process ID shall be removed from the list of process IDs known in the current shell execution environment.

If a process that the shell is waiting for is part of a foreground job that was started as a foreground job and is stopped by a catchable signal (SIGTSTP, SIGTTIN, or SIGTTOU):

- If the currently executing AND-OR list within the list comprising the foreground job consists of a single pipeline in which all of the commands are simple commands, the shell shall either create a suspended job consisting of at least that AND-OR list and the remaining (if any) AND-OR lists in the same list, or create a suspended job consisting of just that AND-OR list and discard the remaining (if any) AND-OR lists in the same list.
- Otherwise, the shell shall create a suspended job consisting of a set of commands, from within the list comprising the foreground job, that is unspecified except that the set shall include at least the pipeline to which the stopped process belongs. Commands in the foreground job that have not already completed and are not included in the suspended job shall be discarded.

Note: Although only a pipeline of simple commands is guaranteed to remain intact if started in the foreground and subsequently suspended, it is possible to ensure that a complex AND-OR list will remain intact when suspended by starting it in the background and immediately bringing it into the foreground. For example:

```
command1 && command2 | { command3 || command4; } & fg
```

If a process that the shell is waiting for is part of a foreground job that was started as a foreground job and is stopped by a SIGSTOP signal, the behavior shall be as described above for a catchable signal unless the shell was executing a built-in utility in the current shell execution environment when the SIGSTOP was delivered, resulting in the shell itself being stopped by the signal, in which case if the shell subsequently receives a SIGCONT signal and has one or more child processes that remain stopped, the shell shall create a suspended job as if only those child processes had been stopped.

When a suspended job is created as a result of a foreground job being stopped, it shall be assigned a job number, and an interactive shell shall write, and a non-interactive shell may write, a message to standard error, formatted as described by the *jobs* utility (without the **–l**

option) for a suspended job. The message may indicate that the commands comprising the job include commands that have already completed; in this case the completed commands shall not be repeated if execution of the job is subsequently continued. If the shell is interactive, it shall save the terminal settings before changing them to the settings it needs to read further commands.

When a process associated with a background job is stopped by a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal, the shell shall convert the (non-suspended) background job into a suspended job and an interactive shell shall write a message to standard error, formatted as described by the *jobs* utility (without the **–l** option) for a suspended job, at the following time:

- If *set* **–b** is enabled, the message shall be written either immediately after the job became suspended or immediately prior to writing the next prompt for input.
- If *set* **–b** is disabled, the message shall be written immediately prior to writing the next prompt for input.

Execution of a suspended job can be continued as a foreground job by means of the *fg* utility (if supported), or as a (non-suspended) background job either by means of the *bg* utility (if supported) or by sending the stopped processes a SIGCONT signal. The *fg* and *bg* utilities shall send a SIGCONT signal to the process group of the process(es) whose stopped wait status caused the shell to suspend the job. If the shell has a controlling terminal, the *fg* utility shall send the SIGCONT signal after it has set the foreground process group ID associated with the terminal (see above). If the *fg* utility is used from an interactive shell to bring into the foreground a suspended job that was created from a foreground job, before it sends the SIGCONT signal the *fg* utility shall restore the terminal settings to the ones that the shell saved when the job was suspended.

When a background job completes or is terminated by a signal, an interactive shell shall write a message to standard error, formatted as described by the *jobs* utility (without the **–l** option) for a job that completed or was terminated by a signal, respectively, at the following time:

- If *set* **–b** is enabled, the message shall be written immediately after the job completes or is terminated.
- If *set* **–b** is disabled, the message shall be written immediately prior to writing the next prompt for input.

In each case above where an interactive shell writes a message immediately prior to writing the next prompt for input, the same message may also be written by a non-interactive shell, at any of the following times:

- After the next time a foreground job terminates or is suspended
- Before the shell parses further input
- Before the shell exits

2.12 Signals and Error Handling

If job control is disabled (see the description of *set* **–m**) when the shell executes an asynchronous AND-OR list, the commands in the list shall inherit from the shell a signal action of ignored (SIG_IGN) for the SIGINT and SIGQUIT signals. In all other cases, commands executed by the shell shall inherit the same signal actions as those inherited by the shell from its parent unless a signal action is modified by the *trap* special built-in (see *trap*)

When a signal for which a trap has been set is received while the shell is waiting for the completion of a utility executing a foreground command, the trap associated with that signal

shall not be executed until after the foreground command has completed. When the shell is waiting, by means of the *wait* utility, for asynchronous commands to complete, the reception of a signal for which a trap has been set shall cause the *wait* utility to return immediately with an exit status >128, immediately after which the trap associated with that signal shall be taken.

If multiple signals are pending for the shell for which there are associated trap actions, the order of execution of trap actions is unspecified.

2.13 Shell Execution Environment

A shell execution environment consists of the following:

- Open files inherited upon invocation of the shell, plus open files controlled by *exec*
- Working directory as set by *cd*
- File creation mask set by *umask*
- File size limit as set by *ulimit*
- Current traps set by *trap*
- Shell parameters that are set by variable assignment (see the *set* special built-in) or from the System Interfaces volume of POSIX.1-2024 environment inherited by the shell when it begins (see the *export* special built-in)
- Shell functions; see Section 2.9.5
- Options turned on at invocation or by *set*
- Background jobs and their associated process IDs, and process IDs of child processes created to execute asynchronous AND-OR lists while job control is disabled; together these process IDs constitute the process IDs ``known to this shell environment''. If the implementation supports non-job-control background jobs, the list of known process IDs and the list of background jobs may form a single list even though this standard describes them as being updated separately. See Section 2.9.3.1
- Shell aliases; see Section 2.3.1

Utilities other than the special built-ins (see Section 2.15) shall be invoked in a separate environment that consists of the following. The initial value of these objects shall be the same as that for the parent shell, except as noted below.

- Open files inherited on invocation of the shell, open files controlled by the *exec* special built-in plus any modifications, and additions specified by any redirections to the utility
- Current working directory
- File creation mask
- If the utility is a shell script, traps caught by the shell shall be set to the default values and traps ignored by the shell shall be set to be ignored by the utility; if the utility is not a shell script, the trap actions (default or ignore) shall be mapped into the appropriate signal handling actions for the utility
- Variables with the *export* attribute, along with those explicitly exported for the duration of the command, shall be passed to the utility environment variables
- It is unspecified whether environment variables that were passed to the invoking shell when it was invoked itself, but were not used to initialize shell variables (see Section 2.5.3) because they had invalid names, are included in the invoked utility's environment.

The environment of the shell process shall not be changed by the utility unless explicitly specified by the utility description (for example, *cd* and *umask*).

A subshell environment shall be created as a duplicate of the shell environment, except that:

- Unless specified otherwise (see *trap*), traps that are not being ignored shall be set to the default action.
- If the shell is interactive, the subshell shall behave as a non-interactive shell in all respects except:
  - — The expansion of the special parameter `'-'` may continue to indicate that it is interactive.
  - — The *set* –**n** option may be ignored.

Changes made to the subshell environment shall not affect the shell environment. Command substitution, commands that are grouped with parentheses, and asynchronous AND-OR lists shall be executed in a subshell environment. Additionally, each command of a multi-command pipeline is in a subshell environment; as an extension, however, any or all commands in a pipeline may be executed in the current environment. Except where otherwise stated, all other commands shall be executed in the current shell environment.

2.14 Pattern Matching Notation

The pattern matching notation described in this section is used to specify patterns for matching character strings in the shell. This notation is also used by some other utilities (*find*, *pax*, and optionally *make*) and by some system interfaces (*fnmatch*( ), *glob*( ), and *wordexp*( )). Historically, pattern matching notation is related to, but slightly different from, the regular expression notation described in XBD Chapter 9. For this reason, the description of the rules for this pattern matching notation are based on the description of regular expression notation, modified to account for the differences.

If an attempt is made to use pattern matching notation to match a string that contains one or more bytes that do not form part of a valid character, the behavior is unspecified. Since pathnames can contain such bytes, portable applications need to ensure that the current locale is the C or POSIX locale when performing pattern matching (or expansion) on arbitrary pathnames.

2.14.1 Patterns Matching a Single Character

The following patterns shall match a single character: ordinary characters, special pattern characters, and pattern bracket expressions. The pattern bracket expression also shall match a single collating element.

In a pattern, or part of one, where a shell-quoting <backslash> can be used, a <backslash> character shall escape the following character as described in Section 2.2.1, regardless of whether or not the <backslash> is inside a bracket expression. (The sequence `"\\"` represents one literal <backslash>.)

In a pattern, or part of one, where a shell-quoting <backslash> cannot be used to preserve the literal value of a character that would otherwise be treated as special:

- A <backslash> character that is not inside a bracket expression shall preserve the literal value of the following character, unless the following character is in a part of the pattern where shell quoting can be used and is a shell quoting character, in which case the

behavior is unspecified.

- For the shell only, it is unspecified whether or not a <backslash> character inside a bracket expression preserves the literal value of the following character.

All of the requirements and effects of quoting on ordinary, shell special, and special pattern characters shall apply to escaping in this context, except where specified otherwise. (Situations where this applies include word expansions when a pattern used in pathname expansion is not present in the original word but results from an earlier expansion, or the argument to the *find* *–name* or *–path* primary as passed to *find*, or the *pattern* argument to the *fnmatch*( ) and *glob*( ) functions when FNM_NOESCAPE or GLOB_NOESCAPE is not set in *flags*, respectively.)

If a pattern ends with an unescaped <backslash>, the behavior is unspecified.

An ordinary character is a pattern that shall match itself. In a pattern, or part of one, where a shell-quoting <backslash> can be used, an ordinary character can be any character in the supported character set except for NUL, those special shell characters in Section 2.2 that require quoting, and the three special pattern characters described below. In a pattern, or part of one, where a shell-quoting <backslash> cannot be used to preserve the literal value of a character that would otherwise be treated as special, an ordinary character can be any character in the supported character set except for NUL and the three special pattern characters described below. Matching shall be based on the bit pattern used for encoding the character, not on the graphic representation of the character. If any character (ordinary, shell special, or pattern special) is quoted, or escaped with a <backslash>, that pattern shall match the character itself. The application shall ensure that it quotes or escapes any character that would otherwise be treated as special, in order for it to be matched as an ordinary character.

When unquoted, unescaped, and not inside a bracket expression, the following three characters shall have special meaning in the specification of patterns:

`?` A <question-mark> is a pattern that shall match any character.

`*` An <asterisk> is a pattern that shall match multiple characters, as described in Section 2.14.2.

`[` A <left-square-bracket> shall introduce a bracket expression if the characters following it meet the requirements for bracket expressions stated in XBD Section 9.3.5, except that the <exclamation-mark> character (`'!'`) shall replace the <circumflex> character (`'^'`) in its role in a non-matching list in the regular expression notation. A bracket expression starting with an unquoted <circumflex> character produces unspecified results. A <left-square-bracket> that does not introduce a valid bracket expression shall match the character itself.

2.14.2 Patterns Matching Multiple Characters

The following rules are used to construct patterns matching multiple characters from patterns matching a single character:

1. The <asterisk> (`'*'`) is a pattern that shall match any string, including the null string.
2. The concatenation of patterns matching a single character is a valid pattern that shall match the concatenation of the single characters or collating elements matched by each of the concatenated patterns.
3. The concatenation of one or more patterns matching a single character with one or more <asterisk> characters is a valid pattern. In such patterns, each <asterisk> shall match a string of zero or more characters, matching the greatest possible number of characters that still allows the remainder of the pattern to match the string.

2.14.3 Patterns Used for Filename Expansion

The rules described so far in Section 2.14.1 and Section 2.14.2 are qualified by the following rules that apply when pattern matching notation is used for filename expansion:

1. The <slash> character in a pathname shall be explicitly matched by using one or more <slash> characters in the pattern; it shall neither be matched by the <asterisk> or <question-mark> special characters nor by a bracket expression. <slash> characters in the pattern shall be identified before bracket expressions; thus, a <slash> cannot be included in a pattern bracket expression used for filename expansion. If a <slash> character is found following an unescaped <left-square-bracket> character before a corresponding <right-square-bracket> is found, the open bracket shall be treated as an ordinary character. For example, the pattern `"a[b/c]d"` does not match such pathnames as **abd** or **a/d**. It only matches a pathname of literally **a[b/c]d**.

2. If a filename begins with a <period> (`'.'`), the <period> shall be explicitly matched by using a <period> as the first character of the pattern or immediately following a <slash> character. The leading <period> shall not be matched by:

   - The <asterisk> or <question-mark> special characters
   - A bracket expression containing a non-matching list, such as `"[!a]"`, a range expression, such as `"[%-0]"`, or a character class expression, such as `"[[:punct:]]"`

   It is unspecified whether an explicit <period> in a bracket expression matching list, such as `"[.abc]"`, can match a leading <period> in a filename.

3. If a specified pattern contains any `'*'`, `'?'` or `'['` characters that will be treated as special (see Section 2.14.1), it shall be matched against existing filenames and pathnames, as appropriate; if directory entries for dot and dot-dot exist, they may be ignored. Each component that contains any such characters shall require read permission in the directory containing that component. Each component that contains a <backslash> that will be treated as special may require read permission in the directory containing that component. Any component, except the last, that does not contain any `'*'`, `'?'` or `'['` characters that will be treated as special shall require search permission. If these permissions are denied, or if an attempt to open or search a pathname as a directory, or an attempt to read an opened directory, fails because of an error condition that is related to file system contents, this shall not be considered an error and pathname expansion shall continue as if the pathname had named an existing directory which had been successfully opened and read, or searched, and no matching directory entries had been found in it. For other error conditions it is unspecified whether pathname expansion fails or they are treated the same as when permission is denied.

   For example, given the pattern:

   ```
   /foo/bar/x*/bam
   ```

   search permission is needed for directories **/** and **foo**, search and read permissions are needed for directory **bar**, and search permission is needed for each **x*** directory.

   If the pattern matches any existing filenames or pathnames, the pattern shall be replaced with those filenames and pathnames, sorted according to the collating sequence in effect in the current locale. If this collating sequence does not have a total ordering of all characters (see XBD Section 7.3.2, on page 139), any filenames or pathnames that collate equally shall be further compared byte-by-byte using the collating sequence for the POSIX locale.

   If the pattern contains an open bracket (`'['`) that does not introduce a bracket expression

as in XBD Section 9.3.5, it is unspecified whether other unquoted `'*'`, `'?'`, `'['` or <backslash> characters within the same slash-delimited component of the pattern retain their special meanings or are treated as ordinary characters. For example, the pattern `"a*[/b*"` may match all filenames beginning with `'b'` in the directory `"a*["` or it may match all filenames beginning with `'b'` in all directories with names beginning with `'a'` and ending with `'['`.

If the pattern does not match any existing filenames or pathnames, the pattern string shall be left unchanged.

Note: A future version of this standard may require that directory entries for dot and dot-dot are ignored (if they exist) when matching patterns against existing filenames. For example, when expanding the pattern `".*"` the result would not include dot and dot-dot.

4. If a specified pattern does not contain any `'*'`, `'?'` or `'['` characters that will be treated as special, the pattern string shall be left unchanged.

2.15 Special Built-In Utilities

The following ``special built-in'' utilities shall be supported in the shell command language. The output of each command, if any, shall be written to standard output, subject to the normal redirection and piping possible with all commands.

The term ``built-in'' implies that there is no need to execute a separate executable file because the utility is implemented in the shell itself. An implementation may choose to make any utility a built-in; however, the special built-in utilities described here differ from regular built-in utilities in two respects:

1. An error in a special built-in utility may cause a shell executing that utility to abort, while an error in a regular built-in utility shall not cause a shell executing that utility to abort. (See Section 2.8.1 for the consequences of errors on interactive and non-interactive shells.) If a special built-in utility encountering an error does not abort the shell, its exit value shall be non-zero.

2. As described in Section 2.9.1, variable assignments preceding the invocation of a special built-in utility affect the current execution environment; this shall not be the case with a regular built-in or other utility.

The special built-in utilities in this section need not be provided in a manner accessible via the *exec* family of functions defined in the System Interfaces volume of POSIX.1-2024.

Some of the special built-ins are described as conforming to XBD Section 12.2. For those that are not, the requirement in Section 1.4 that `"--"` be recognized as a first argument to be discarded does not apply and a conforming application shall not use that argument.

82359 **NAME**

82360 break — exit from for, while, or until loop

82361 **SYNOPSIS**

82362 `break [n]`

82363 **DESCRIPTION**

82364 If *n* is specified, the *break* utility shall exit from the *n*th enclosing **for**, **while**, or **until** loop. If *n* is
82365 not specified, *break* shall behave as if *n* was specified as 1. Execution shall continue with the
82366 command immediately following the exited loop. The application shall ensure that the value of
82367 *n* is a positive decimal integer. If *n* is greater than the number of enclosing loops, the outermost
82368 enclosing loop shall be exited. If there is no enclosing loop, the behavior is unspecified.

82369 A loop shall enclose a *break* or *continue* command if the loop lexically encloses the command. A
82370 loop lexically encloses a *break* or *continue* command if the command is:

82371 • Executing in the same execution environment (see Section 2.13) as the compound-list of the
82372 loop's do-group (see Section 2.10.2), and

82373 • Contained in a compound-list associated with the loop (either in the compound-list of the
82374 loop's do-group or, if the loop is a **while** or **until** loop, in the compound-list following the
82375 **while** or **until** reserved word), and

82376 • Not in the body of a function whose function definition command (see Section 2.9.5) is
82377 contained in a compound-list associated with the loop.

82378 If *n* is greater than the number of lexically enclosing loops and there is a non-lexically enclosing
82379 loop in progress in the same execution environment as the *break* or *continue* command, it is
82380 unspecified whether that loop encloses the command.

82381 **OPTIONS**

82382 None.

82383 **OPERANDS**

82384 See the DESCRIPTION.

82385 **STDIN**

82386 Not used.

82387 **INPUT FILES**

82388 None.

82389 **ENVIRONMENT VARIABLES**

82390 None.

82391 **ASYNCHRONOUS EVENTS**

82392 Default.

82393 **STDOUT**

82394 Not used.

82395 **STDERR**

82396 The standard error shall be used only for diagnostic messages.

82397 **OUTPUT FILES**

82398 None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

0 Successful completion.

>0 The *n* value was not an unsigned decimal integer greater than or equal to 1.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

```
for i in *
do
    if test -d "$i"
    then break
    fi
done
```

The results of running the following example are unspecified: there are two loops in progress when the *break* command is executed, and they are in the same execution environment, but neither loop is lexically enclosing the *break* command. (There are no loops lexically enclosing the *continue* commands, either.)

```
foo() {
    for j in 1 2; do
        echo 'break 2' >/tmp/do_break
        echo "  sourcing /tmp/do_break ($j)..."
        # the behavior of the break from running the following command
        # results in unspecified behavior:
        . /tmp/do_break

        do_continue() { continue 2; }
        echo "  running do_continue ($j)..."
        # the behavior of the continue in the following function call
        # results in unspecified behavior (if execution reaches this
        # point):
        do_continue

        trap 'continue 2' USR1
        echo "  sending SIGUSR1 to self ($j)..."
        # the behavior of the continue in the trap invoked from the
        # following signal results in unspecified behavior (if
        # execution reaches this point):
        kill -s USR1 $$
        sleep 1
    done
}
for i in 1 2; do
    echo "running foo ($i)..."
    foo
done
```


RATIONALE

In early proposals, consideration was given to expanding the syntax of *break* and *continue* to refer to a label associated with the appropriate loop as a preferable alternative to the *n* method. However, this volume of POSIX.1-2024 does reserve the name space of command names ending with a <colon>. It is anticipated that a future implementation could take advantage of this and provide something like:

```
outofloop: for i in a b c d e
do
    for j in 0 1 2 3 4 5 6 7 8 9
    do
        if test -r "${i}${j}"
        then break outofloop
        fi
    done
done
```

and that this might be standardized after implementation experience is achieved.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.15

CHANGE HISTORY

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page sections use terms as described in the Utility Description Defaults (Section 1.4). No change in behavior is intended.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0046 [842] is applied.

Issue 8

Austin Group Defect 1058 is applied, clarifying that the requirement for *n* to be a positive decimal integer is a requirement on the application.

82475 **NAME**

82476 colon — null utility

82477 **SYNOPSIS**

82478 `: [`*`argument`*`...]`

82479 **DESCRIPTION**

82480 This utility shall do nothing except return a 0 exit status. It is used when a command is needed,
82481 as in the **then** condition of an **if** command, but nothing is to be done by the command.

82482 **OPTIONS**

82483 This utility shall not recognize the `"--"` argument in the manner specified by Guideline 10 of
82484 XBD Section 12.2 (on page 215).

82485 Implementations shall not support any options.

82486 **OPERANDS**

82487 See the DESCRIPTION.

82488 **STDIN**

82489 Not used.

82490 **INPUT FILES**

82491 None.

82492 **ENVIRONMENT VARIABLES**

82493 None.

82494 **ASYNCHRONOUS EVENTS**

82495 Default.

82496 **STDOUT**

82497 Not used.

82498 **STDERR**

82499 Not used.

82500 **OUTPUT FILES**

82501 None.

82502 **EXTENDED DESCRIPTION**

82503 None.

82504 **EXIT STATUS**

82505 Zero.

82506 **CONSEQUENCES OF ERRORS**

82507 None.

82508 **APPLICATION USAGE**

82509 See the APPLICATION USAGE for *true*.

82510 **EXAMPLES**

82511 ```
: "${X=abc}"
if   false
then   :
else   printf '%s\n' "$X"
fi
```
82516 **abc**

82517 As with any of the special built-ins, the null utility can also have variable assignments and

  
```

82518 redirections associated with it, such as:

82519 `x=y : > z`

82520 which sets variable *x* to the value *y* (so that it persists after the null utility completes) and creates
82521 or truncates file **z**; if the file cannot be created or truncated, a non-interactive shell exits (see
82522 Section 2.8.1).

82523 **RATIONALE**

82524 None.

82525 **FUTURE DIRECTIONS**

82526 None.

82527 **SEE ALSO**

82528 Section 2.15, *true*

82529 **CHANGE HISTORY**

82530 **Issue 6**

82531 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
82532 sections use terms as described in the Utility Description Defaults (Section 1.4). No change in
82533 behavior is intended.

82534 **Issue 7**

82535 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

82536 **Issue 8**

82537 Austin Group Defect 1272 is applied, clarifying that the null utility does not process its
82538 arguments, does not recognize the `"--"` end-of-options delimiter, does not support any options,
82539 and does not write to standard error.

82540 Austin Group Defect 1640 is applied, changing the APPLICATION USAGE section.

82541 **NAME**

82542 continue — continue for, while, or until loop

82543 **SYNOPSIS**

82544 `continue [n]`

82545 **DESCRIPTION**

82546 If *n* is specified, the *continue* utility shall return to the top of the *n*th enclosing **for**, **while**, or **until**
82547 loop. If *n* is not specified, *continue* shall behave as if *n* was specified as 1. Returning to the top of
82548 the loop involves repeating the condition list of a **while** or **until** loop or performing the next
82549 assignment of a **for** loop, and re-executing the loop if appropriate.

82550 The application shall ensure that the value of *n* is a positive decimal integer. If *n* is greater than
82551 the number of enclosing loops, the outermost enclosing loop shall be used. If there is no
82552 enclosing loop, the behavior is unspecified.

82553 The meaning of ``enclosing'' shall be as specified in the description of the *break* utility.

82554 **OPTIONS**

82555 None.

82556 **OPERANDS**

82557 See the DESCRIPTION.

82558 **STDIN**

82559 Not used.

82560 **INPUT FILES**

82561 None.

82562 **ENVIRONMENT VARIABLES**

82563 None.

82564 **ASYNCHRONOUS EVENTS**

82565 Default.

82566 **STDOUT**

82567 Not used.

82568 **STDERR**

82569 The standard error shall be used only for diagnostic messages.

82570 **OUTPUT FILES**

82571 None.

82572 **EXTENDED DESCRIPTION**

82573 None.

82574 **EXIT STATUS**

82575 0 Successful completion.

82576 >0 The *n* value was not an unsigned decimal integer greater than or equal to 1.

82577 **CONSEQUENCES OF ERRORS**

82578 Default.

APPLICATION USAGE

None.

EXAMPLES

```
for i in *
do
    if test -d "$i"
    then continue
    fi
    printf '"%s" is not a directory.\n' "$i"
done
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.15

CHANGE HISTORY

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page sections use terms as described in the Utility Description Defaults (Section 1.4). No change in behavior is intended.

Issue 7

The example is changed to use the *printf* utility rather than *echo*.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0046 [842] is applied.

Issue 8

Austin Group Defect 1058 is applied, clarifying that the requirement for *n* to be a positive decimal integer is a requirement on the application.

82606 **NAME**

82607 dot — execute commands in the current environment

82608 **SYNOPSIS**

82609 `. file`

82610 **DESCRIPTION**

82611 The shell shall tokenize (see Section 2.3) the contents of the *file*, parse the tokens (see Section
82612 2.10), and execute the resulting commands in the current environment. It is unspecified whether
82613 the commands are parsed and executed as a *program* (as for a shell script) or are parsed as a
82614 single *compound_list* that is executed after the entire file has been parsed.

82615 If *file* does not contain a <slash>, the shell shall use the search path specified by *PATH* to find the
82616 directory containing *file*. Unlike normal command search, however, the file searched for by the
82617 *dot* utility need not be executable. If no readable file is found, a non-interactive shell shall abort;
82618 an interactive shell shall write a diagnostic message to standard error.

82619 The *dot* special built-in shall support XBD Section 12.2 (on page 215), except for Guidelines 1 and
82620 2.

82621 **OPTIONS**

82622 None.

82623 **OPERANDS**

82624 See the DESCRIPTION.

82625 **STDIN**

82626 Not used.

82627 **INPUT FILES**

82628 See the DESCRIPTION.

82629 **ENVIRONMENT VARIABLES**

82630 See the DESCRIPTION.

82631 **ASYNCHRONOUS EVENTS**

82632 Default.

82633 **STDOUT**

82634 Not used.

82635 **STDERR**

82636 The standard error shall be used only for diagnostic messages.

82637 **OUTPUT FILES**

82638 None.

82639 **EXTENDED DESCRIPTION**

82640 None.

82641 **EXIT STATUS**

82642 If no readable file was found or if the commands in the file could not be parsed, and the shell is
82643 interactive (and therefore does not abort; see Section 2.8.1), the exit status shall be non-zero.
82644 Otherwise, return the value of the last command executed, or a zero exit status if no command is
82645 executed.

82646 **CONSEQUENCES OF ERRORS**

82647 Default.

APPLICATION USAGE

None.

EXAMPLES

```
cat foobar
foo=hello bar=world
. ./foobar
echo $foo $bar
hello world
```

RATIONALE

Some older implementations searched the current directory for the *file*, even if the value of *PATH* disallowed it. This behavior was omitted from this volume of POSIX.1-2024 due to concerns about introducing the susceptibility to trojan horses that the user might be trying to avoid by leaving **dot** out of *PATH*.

The KornShell version of *dot* takes optional arguments that are set to the positional parameters. This is a valid extension that allows a *dot* script to behave identically to a function.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.15, *return*

CHANGE HISTORY

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page sections use terms as described in the Utility Description Defaults (Section 1.4). No change in behavior is intended.

Issue 7

SD5-XCU-ERN-164 is applied.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0038 [114] and XCU/TC1-2008/0039 [214] are applied.

Issue 8

Austin Group Defect 252 is applied, adding a requirement for *dot* to support XBD Section 12.2 (except for Guidelines 1 and 2, since the utility's name is `'.'`).

Austin Group Defect 953 is applied, clarifying how the commands in the *file* are parsed.

Austin Group Defect 1265 is applied, updating the DESCRIPTION to align with the changes made to Section 2.8.1 between Issue 6 and Issue 7.

82682 **NAME**

82683 eval — construct command by concatenating arguments

82684 **SYNOPSIS**

82685 `eval [`*`argument`*`...]`

82686 **DESCRIPTION**

82687 The *eval* utility shall construct a command string by concatenating *argument*s together,
82688 separating each with a <space> character. The constructed command string shall be tokenized
82689 (see Section 2.3), parsed (see Section 2.10), and executed by the shell in the current environment.
82690 It is unspecified whether the commands are parsed and executed as a *program* (as for a shell
82691 script) or are parsed as a single *compound_list* that is executed after the entire constructed
82692 command string has been parsed.

82693 **OPTIONS**

82694 None.

82695 **OPERANDS**

82696 See the DESCRIPTION.

82697 **STDIN**

82698 Not used.

82699 **INPUT FILES**

82700 None.

82701 **ENVIRONMENT VARIABLES**

82702 None.

82703 **ASYNCHRONOUS EVENTS**

82704 Default.

82705 **STDOUT**

82706 Not used.

82707 **STDERR**

82708 The standard error shall be used only for diagnostic messages.

82709 **OUTPUT FILES**

82710 None.

82711 **EXTENDED DESCRIPTION**

82712 None.

82713 **EXIT STATUS**

82714 If there are no *argument*s, or only null *argument*s, *eval* shall return a zero exit status; otherwise, it
82715 shall return the exit status of the command defined by the string of concatenated *argument*s
82716 separated by <space> characters, or a non-zero exit status if the concatenation could not be
82717 parsed as a command and the shell is interactive (and therefore did not abort).

82718 **CONSEQUENCES OF ERRORS**

82719 Default.

APPLICATION USAGE

Since *eval* is not required to recognize the `"--"` end of options delimiter, in cases where the argument(s) to *eval* might begin with `'-'` it is recommended that the first argument is prefixed by a string that will not alter the commands to be executed, such as a <space> character:

```
eval " $commands"
```

or:

```
eval " $(some_command)"
```

EXAMPLES

```
foo=10 x=foo
y='$'$x
echo $y
$foo
eval y='$'$x
echo $y
10
```

RATIONALE

This standard allows, but does not require, *eval* to recognize `"--"`. Although this means applications cannot use `"--"` to protect against options supported as an extension (or errors reported for unsupported options), the nature of the *eval* utility is such that other means can be used to provide this protection (see APPLICATION USAGE above).

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.15

CHANGE HISTORY

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page sections use terms as described in the Utility Description Defaults (Section 1.4). No change in behavior is intended.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0040 [114], XCU/TC1-2008/0041 [163], and XCU/TC1-2008/0042 [163] are applied.

Issue 8

Austin Group Defect 953 is applied, clarifying how the commands in the constructed command string are parsed.

NAME

exec — perform redirections in the current shell or execute a utility

SYNOPSIS

```
exec [utility [argument...]]
```

DESCRIPTION

If *exec* is specified with no operands, any redirections associated with the *exec* command shall be made in the current shell execution environment. If any file descriptors with numbers greater than 2 are opened by those redirections, it is unspecified whether those file descriptors remain open when the shell invokes another utility. Scripts concerned that child shells could misuse open file descriptors can always close them explicitly, as shown in one of the following examples. If the result of the redirections would be that file descriptor 0, 1, or 2 is closed, implementations may open the file descriptor to an unspecified file.

If *exec* is specified with a *utility* operand, the shell shall execute a non-built-in utility as described in Section 2.9.1.6 with *utility* as the command name and the *argument* operands (if any) as the command arguments.

If the *exec* command fails, a non-interactive shell shall exit from the current shell execution environment; an interactive shell may exit from a subshell environment but shall not exit if the current shell environment is not a subshell environment.

If the *exec* command fails and the shell does not exit, any redirections associated with the *exec* command that were successfully made shall take effect in the current shell execution environment.

The *exec* special built-in shall support XBD Section 12.2 (on page 215).

OPTIONS

None.

OPERANDS

See the DESCRIPTION.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variable shall affect the execution of *exec*:

PATH Determine the search path when looking for the utility given as the *utility* operand; see XBD Section 8.3 (on page 174).

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

If *utility* is specified and is executed, *exec* shall not return to the shell; rather, the exit status of the current shell execution environment shall be the exit status of *utility*. If *utility* is specified and an attempt to execute it as a non-built-in utility fails, the exit status shall be as described in Section 2.9.1.6. If a redirection error occurs (see Section 2.8.1), the exit status shall be a value in the range 1–125. Otherwise, *exec* shall return a zero exit status.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

Open *readfile* as file descriptor 3 for reading:

```
exec 3< readfile
```

Open *writefile* as file descriptor 4 for writing:

```
exec 4> writefile
```

Make file descriptor 5 a copy of file descriptor 0:

```
exec 5<&0
```

Close file descriptor 3:

```
exec 3<&-
```

Cat the file **maggie** by replacing the current shell with the *cat* utility:

```
exec cat maggie
```

An application that is not concerned with strict conformance can make use of optional `%g` support known to be present in the implementation's *printf* utility by ensuring that any shell built-in version is not executed instead, and using a subshell so that the shell continues afterwards:

```
(exec printf '%g\n' "$float_value")
```

RATIONALE

Most historical implementations were not conformant in that:

```
foo=bar exec cmd
```

did not pass **foo** to **cmd**.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.15

CHANGE HISTORY

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page sections use terms as described in the Utility Description Defaults (Section 1.4). No change in behavior is intended.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 252 is applied, adding a requirement for *exec* to support XBD Section 12.2.

Austin Group Defect 1157 is applied, clarifying the execution of non-built-in utilities.

Austin Group Defect 1587 is applied, changing the ENVIRONMENT VARIABLES section.

NAME

exit — cause the shell to exit

SYNOPSIS

```
exit [n]
```

DESCRIPTION

The *exit* utility shall cause the shell to exit from its current execution environment. If the current execution environment is a subshell environment, the shell shall exit from the subshell environment and continue in the environment from which that subshell environment was invoked; otherwise, the shell utility shall terminate. The wait status of the shell or subshell shall be determined by the unsigned decimal integer *n*, if specified.

If *n* is specified and has a value between 0 and 255 inclusive, the wait status of the shell or subshell shall indicate that it exited with exit status *n*. If *n* is specified and has a value greater than 256 that corresponds to an exit status the shell assigns to commands terminated by a valid signal (see Section 2.8.2), the wait status of the shell or subshell shall indicate that it was terminated by that signal. No other actions associated with the signal, such as execution of *trap* actions or creation of a core image, shall be performed by the shell.

If *n* is specified and is not an unsigned decimal integer, or has a value of 256, or has a value greater than 256 but not corresponding to an exit status the shell assigns to commands terminated by a valid signal, the wait status of the shell or subshell is unspecified.

If *n* is not specified, the result shall be as if *n* were specified with the current value of the special parameter `'?'` (see Section 2.5.2), except that if the *exit* command would cause the end of execution of a *trap* action, the value for the special parameter `'?'` that is considered ``current'' shall be the value it had immediately preceding the *trap* action.

A *trap* action on **EXIT** shall be executed before the shell terminates, except when the *exit* utility is invoked in that *trap* action itself, in which case the shell shall exit immediately. It is unspecified whether setting a new *trap* action on **EXIT** during execution of a *trap* action on **EXIT** will cause the new *trap* action to be executed before the shell terminates.

OPTIONS

None.

OPERANDS

See the DESCRIPTION.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

82888 **OUTPUT FILES**

82889 None.

82890 **EXTENDED DESCRIPTION**

82891 None.

82892 **EXIT STATUS**

82893 The *exit* utility causes the shell to exit from its current execution environment, and therefore does
82894 not itself return an exit status.

82895 **CONSEQUENCES OF ERRORS**

82896 Default.

82897 **APPLICATION USAGE**

82898 As explained in other sections, certain exit status values have been reserved for special uses and
82899 should be used by applications only for those purposes:

82900 126 A file to be executed was found, but it was not an executable utility.

82901 127 A utility to be executed was not found.

82902 128 An unrecoverable read error was detected by the shell while reading commands, except
82903 from the *file* operand of the *dot* special built-in.

82904 >128 A command was interrupted by a signal.

82905 **EXAMPLES**

82906 Exit with a *true* value:

82907
```
exit 0
```

82908 Exit with a *false* value:

82909
```
exit 1
```

82910 Propagate error handling from within a subshell:

82911
```
(
    command1 || exit 1
    command2 || exit 1
    exec command3
) > outputfile || exit 1
echo "outputfile created successfully"
```

82917 **RATIONALE**

82918 The behavior of *exit* when given an invalid argument or unknown option is unspecified, because
82919 of differing practices in the various historical implementations. A value larger than 255 might be
82920 truncated by the shell, and be unavailable even to a parent process that uses *waitid*() to get the
82921 full exit value. It is recommended that implementations that detect any usage error should cause
82922 a non-zero exit status (or, if the shell is interactive and the error does not cause the shell to abort,
82923 store a non-zero value in "$?"), but even this was not done historically in all shells.

82924 See also Section C.2.8.2 (on page 3894).

82925 **FUTURE DIRECTIONS**

82926 None.

SEE ALSO

Section 2.15

CHANGE HISTORY

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page sections use terms as described in the Utility Description Defaults (Section 1.4). No change in behavior is intended.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0047 [717], XCU/TC2-2008/0048 [960], XCU/TC2-2008/0049 [717], and XCU/TC2-2008/0050 [960] are applied.

Issue 8

Austin Group Defect 51 is applied, specifying the behavior when *n* has a value greater than 256 that corresponds to an exit status the shell assigns to commands terminated by a valid signal.

Austin Group Defect 1029 is applied, changing ``*trap*'' to ``*trap* action'' in the DESCRIPTION section.

Austin Group Defect 1309 is applied, changing the EXIT STATUS section.

Austin Group Defect 1425 is applied, clarifying the requirements for a *trap* action on **EXIT**.

Austin Group Defect 1602 is applied, clarifying the behavior of *exit* in a *trap* action.

Austin Group Defect 1629 is applied, adding exit status 128 to the APPLICATION USAGE section.

82947 **NAME**

82948 export — set the export attribute for variables

82949 **SYNOPSIS**

82950 `export name[=word]...`

82951 `export -p`

82952 **DESCRIPTION**

82953 The shell shall give the *export* attribute to the variables corresponding to the specified *name*s,
82954 which shall cause them to be in the environment of subsequently executed commands. If the
82955 name of a variable is followed by =*word*, then the value of that variable shall be set to *word*.

82956 The *export* special built-in shall be a declaration utility. Therefore, if *export* is recognized as the
82957 command name of a simple command, then subsequent words of the form *name*=*word* shall be
82958 expanded in an assignment context. See Section 2.9.1.1.

82959 The *export* special built-in shall support XBD Section 12.2.

82960 When **−p** is specified, *export* shall write to the standard output the names and values of all
82961 exported variables, in the following format:

82962 `"export %s=%s\n", <name>, <value>`

82963 if *name* is set, and:

82964 `"export %s\n", <name>`

82965 if *name* is unset.

82966 The shell shall format the output, including the proper use of quoting, so that it is suitable for
82967 reinput to the shell as commands that achieve the same exporting results, except:

82968 1. Read-only variables with values cannot be reset.

82969 2. Variables that were unset at the time they were output need not be reset to the unset state
82970 if a value is assigned to the variable between the time the state was saved and the time at
82971 which the saved output is reinput to the shell.

82972 When no arguments are given, the results are unspecified.

82973 **OPTIONS**

82974 See the DESCRIPTION.

82975 **OPERANDS**

82976 See the DESCRIPTION.

82977 **STDIN**

82978 Not used.

82979 **INPUT FILES**

82980 None.

82981 **ENVIRONMENT VARIABLES**

82982 None.

82983 **ASYNCHRONOUS EVENTS**

82984 Default.

STDOUT

See the DESCRIPTION.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

0 Successful completion.

>0 At least one operand could not be processed as requested, such as a *name* operand that could not be exported or an attempt to modify a *readonly* variable using a *name=word* operand, or the **−p** option was specified and a write error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Note that, unless *X* was previously marked readonly, the value of `"$?"` after:

```
export X=$(false)
```

will be 0 (because *export* successfully set *X* to the empty string) and that execution continues, even if *set* **−e** is in effect. In order to detect command substitution failures, a user must separate the assignment from the export, as in:

```
X=$(false)
export X
```

In shells that support extended assignment syntax, for example to allow an array to be populated with a single assignment, such extensions can typically only be used in assignments specified as arguments to *export* if the command word is literally *export*, and not if it is some other word that expands to *export*. For example:

```
# Shells that support array assignment as an extension generally
# support this:
export x=(1 2 3); echo ${x[0]}  # outputs 1
# But generally do not support this:
e=export; $e x=(1 2 3); echo ${x[0]}  # syntax error
```

EXAMPLES

Export *PWD* and *HOME* variables:

```
export PWD HOME
```

Set and export the *PATH* variable:

```
export PATH="/local/bin:$PATH"
```

Save and restore all exported variables:

```
export -p > temp-file
unset a lot of variables
... processing
. ./temp-file
```


Note: If LANG, LC_CTYPE or LC_ALL are left altered or unset in the above example prior to sourcing `temp-file`, the results may be undefined.

RATIONALE

Some historical shells use the no-argument case as the functional equivalent of what is required here with **−p**. This feature was left unspecified because it is not historical practice in all shells, and some scripts may rely on the now-unspecified results on their implementations. Attempts to specify the **−p** output as the default case were unsuccessful in achieving consensus. The **−p** option was added to allow portable access to the values that can be saved and then later restored using; for example, a *dot* script.

Some implementations extend the shell's assignment syntax, for example to allow an array to be populated with a single assignment, and in order for such an extension to be usable in assignments specified as arguments to *export* these shells have *export* as a separate token in their grammar. This standard only permits an extension of this nature when the input to the shell would contain a syntax error according to the standard grammar. Note that although *export* can be a separate token in the shell's grammar, it cannot be a reserved word since *export* is a candidate for alias substitution whereas reserved words are not (see Section 2.3.1).

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.9.1.1, Section 2.15

XBD Section 12.2

CHANGE HISTORY

Issue 6

IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page sections use terms as described in the Utility Description Defaults (Section 1.4). No change in behavior is intended.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/6 is applied, adding the following text to the end of the first paragraph of the DESCRIPTION: ``If the name of a variable is followed by *=word*, then the value of that variable shall be set to *word*.''. The reason for this change is that the SYNOPSIS for *export* includes:

`export name`**`[`**`=`*`word`***`]...`**

but the meaning of the optional ``*=word*'' is never explained in the text.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0043 [352] is applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0051 [654] and XCU/TC2-2008/0052 [960] are applied.

Issue 8

Austin Group Defect 351 is applied, requiring *export* to be a declaration utility.

Austin Group Defect 367 is applied, changing the EXIT STATUS section.

Austin Group Defect 1258 is applied, changing the EXAMPLES section.

83068 Austin Group Defect 1393 is applied, changing the APPLICATION USAGE and RATIONALE
83069 sections.

NAME

readonly — set the readonly attribute for variables

SYNOPSIS

```
readonly name[=word]...
readonly -p
```

DESCRIPTION

The variables whose *name*s are specified shall be given the *readonly* attribute. The values of variables with the *readonly* attribute cannot be changed by subsequent assignment or use of the *export*, *getopts*, *readonly*, or *read* utilities, nor can those variables be unset by the *unset* utility. As described in XBD Section 8.1, conforming applications shall not request to mark a variable as *readonly* if it is documented as being manipulated by a shell built-in utility, as it may render those utilities unable to complete successfully. If the name of a variable is followed by =*word*, then the value of that variable shall be set to *word*.

The *readonly* special built-in shall be a declaration utility. Therefore, if *readonly* is recognized as the command name of a simple command, then subsequent words of the form *name*=*word* shall be expanded in an assignment context. See Section 2.9.1.1.

The *readonly* special built-in shall support XBD Section 12.2.

When **−p** is specified, *readonly* writes to the standard output the names and values of all read-only variables, in the following format:

```
"readonly %s=%s\n", <name>, <value>
```

if *name* is set, and

```
"readonly %s\n", <name>
```

if *name* is unset.

The shell shall format the output, including the proper use of quoting, so that it is suitable for reinput to the shell as commands that achieve the same value and *readonly* attribute-setting results in a shell execution environment in which:

1. Variables with values at the time they were output do not have the *readonly* attribute set.
2. Variables that were unset at the time they were output do not have a value at the time at which the saved output is reinput to the shell.

When no arguments are given, the results are unspecified.

OPTIONS

See the DESCRIPTION.

OPERANDS

See the DESCRIPTION.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

83110 **ASYNCHRONOUS EVENTS**
83111 Default.

83112 **STDOUT**
83113 See the DESCRIPTION.

83114 **STDERR**
83115 The standard error shall be used only for diagnostic messages.

83116 **OUTPUT FILES**
83117 None.

83118 **EXTENDED DESCRIPTION**
83119 None.

83120 **EXIT STATUS**

83121 0 Successful completion.

83122 >0 At least one operand could not be processed as requested, such as a *name* operand that
83123 could not be marked *readonly* or an attempt to modify an already *readonly* variable using a
83124 *name=word* operand, or the **−p** option was specified and a write error occurred.

83125 **CONSEQUENCES OF ERRORS**
83126 Default.

83127 **APPLICATION USAGE**
83128 In shells that support extended assignment syntax, for example to allow an array to be
83129 populated with a single assignment, such extensions can typically only be used in assignments
83130 specified as arguments to *readonly* if the command word is literally *readonly*, and not if it is some
83131 other word that expands to *readonly*. For example:

```
83132 # Shells that support array assignment as an extension generally
83133 # support this:
83134 readonly x=(1 2 3); echo ${x[0]}  # outputs 1
83135 # But generally do not support this:
83136 r=readonly; $r x=(1 2 3); echo ${x[0]}  # syntax error
```

83137 **EXAMPLES**

```
83138 readonly HOME
```

83139 **RATIONALE**
83140 Some historical shells preserve the *readonly* attribute across separate invocations. This volume of
83141 POSIX.1-2024 allows this behavior, but does not require it.

83142 The **−p** option allows portable access to the values that can be saved and then later restored
83143 using, for example, a *dot* script. Also see the RATIONALE for *export* for a description of the no-
83144 argument and **−p** output cases and a related example.

83145 Read-only functions were considered, but they were omitted as not being historical practice or
83146 particularly useful. Furthermore, functions must not be read-only across invocations to preclude
83147 ``spoofing'' (spoofing is the term for the practice of creating a program that acts like a well-
83148 known utility with the intent of subverting the real intent of the user) of administrative or
83149 security-relevant (or security-conscious) shell scripts.

83150 Attempts to set the *readonly* attribute on certain variables, such as *PWD*, may have surprising
83151 results. Either *readonly* will reject the attempt, or the attempt will succeed but the shell will
83152 continue to alter the contents of *PWD* during the *cd* utility, or the attempt will succeed and
83153 render the *cd* utility inoperative (since it must not change directories if it cannot also update

PWD).

Some implementations extend the shell's assignment syntax, for example to allow an array to be populated with a single assignment, and in order for such an extension to be usable in assignments specified as arguments to *readonly* these shells have *readonly* as a separate token in their grammar. This standard only permits an extension of this nature when the input to the shell would contain a syntax error according to the standard grammar. Note that although *readonly* can be a separate token in the shell's grammar, it cannot be a reserved word since *readonly* is a candidate for alias substitution whereas reserved words are not (see Section 2.3.1).

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.9.1.1, Section 2.15

XBD Section 12.2

CHANGE HISTORY

Issue 6

IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page sections use terms as described in the Utility Description Defaults (Section 1.4). No change in behavior is intended.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/7 is applied, adding the following text to the end of the first paragraph of the DESCRIPTION: ``If the name of a variable is followed by *=word*, then the value of that variable shall be set to *word*.''. The reason for this change is that the SYNOPSIS for *readonly* includes:

```
readonly name[=word]...
```

but the meaning of the optional ``*=word*'' is never explained in the text.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0052 [960] is applied.

Issue 8

Austin Group Defect 351 is applied, requiring *readonly* to be a declaration utility.

Austin Group Defect 367 is applied, clarifying that the values of *readonly* variables cannot be changed by subsequent use of the *export*, *getopts*, *readonly*, or *read* utilities, and changing the EXIT STATUS, EXAMPLES and RATIONALE sections.

Austin Group Defect 1393 is applied, changing the APPLICATION USAGE and RATIONALE sections.

NAME

return — return from a function or dot script

SYNOPSIS

```
return [n]
```

DESCRIPTION

The *return* utility shall cause the shell to stop executing the current function or *dot* script. If the shell is not currently executing a function or *dot* script, the results are unspecified.

OPTIONS

None.

OPERANDS

See the DESCRIPTION.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The exit status shall be *n*, if specified, except that the behavior is unspecified if *n* is not an unsigned decimal integer or is greater than 255. If *n* is not specified, the result shall be as if *n* were specified with the current value of the special parameter `'?'` (see Section 2.5.2), except that if the *return* command would cause the end of execution of a *trap* action, the value for the special parameter `'?'` that is considered ``current'' shall be the value it had immediately preceding the *trap* action.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

The behavior of *return* when not in a function or *dot* script differs between the System V shell and the KornShell. In the System V shell this is an error, whereas in the KornShell, the effect is the same as *exit*.

The results of returning a number greater than 255 are undefined because of differing practices in the various historical implementations. Some shells AND out all but the low-order 8 bits; others allow larger values, but not of unlimited size.

See the discussion of appropriate exit status values under *exit*.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.9.5, Section 2.15, *dot*

CHANGE HISTORY

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page sections use terms as described in the Utility Description Defaults (Section 1.4). No change in behavior is intended.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0044 [214] and XCU/TC1-2008/0045 [214] are applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0052 [960] is applied.

Issue 8

Austin Group Defect 1309 is applied, changing the EXIT STATUS section.

Austin Group Defect 1602 is applied, clarifying the behavior of *return* in a *trap* action.

NAME

set — set or unset options and positional parameters

SYNOPSIS

```
set [-abCefhmnuvx] [-o option] [argument...]
set [+abCefhmnuvx] [+o option] [argument...]
set -- [argument...]
set -o
set +o
```

DESCRIPTION

If no *option*s or *argument*s are specified, *set* shall write the names and values of all shell variables in the collation sequence of the current locale. Each *name* shall start on a separate line, using the format:

```
"%s=%s\n", <name>, <value>
```

The *value* string shall be written with appropriate quoting; see the description of shell quoting in Section 2.2. The output shall be suitable for reinput to the shell, setting or resetting, as far as possible, the variables that are currently set; read-only variables cannot be reset.

When options are specified, they shall set or unset attributes of the shell, as described below. When *argument*s are specified, they cause positional parameters to be set or unset, as described below. Setting or unsetting attributes and positional parameters are not necessarily related actions, but they can be combined in a single invocation of *set*.

The *set* special built-in shall support XBD Section 12.2 except that options can be specified with either a leading <hyphen-minus> (meaning enable the option) or <plus-sign> (meaning disable it) unless otherwise specified.

Implementations shall support the options in the following list in both their <hyphen-minus> and <plus-sign> forms. These options can also be specified as options to *sh*.

−a Set the *export* attribute for all variable assignments. When this option is on, whenever a value is assigned to a variable in the current shell execution environment, the *export* attribute shall be set for the variable. This applies to all forms of assignment, including those made as a side-effect of variable expansions or arithmetic expansions, and those made as a result of the operation of the *cd*, *getopts*, or *read* utilities.

> **Note:** As discussed in Section 2.9.1, not all variable assignments happen in the current execution environment. When an assignment happens in a separate execution environment the *export* attribute is still set for the variable, but that does not affect the current execution environment.

−b This option shall be supported if the implementation supports the User Portability Utilities option. When job control and **−b** are both enabled, the shell shall write asynchronous notifications of background job completions (including termination by a signal), and may write asynchronous notifications of background job suspensions. See Section 2.11 for details. When job control is disabled, the **−b** option shall have no effect. Asynchronous notification shall not be enabled by default.

−C (Uppercase C.) Prevent existing regular files from being overwritten by the shell's `'>'` redirection operator (see Section 2.7.2); the `">|"` redirection operator shall override this *noclobber* option for an individual file.

−e When this option is on, when any command fails (for any of the reasons listed in Section 2.8.1 or by returning an exit status greater than zero), the shell immediately shall exit, as if by executing the *exit* special built-in utility with no arguments, with the following exceptions:

1. The failure of any individual command in a multi-command pipeline, or of any subshell environments in which command substitution was performed during word expansion, shall not cause the shell to exit. Only the failure of the pipeline itself shall be considered.
2. The **−e** setting shall be ignored when executing the compound list following the **while**, **until**, **if**, or **elif** reserved word, a pipeline beginning with the **!** reserved word, or any command of an AND-OR list other than the last.
3. If the exit status of a compound command other than a subshell command was the result of a failure while **−e** was being ignored, then **−e** shall not apply to this command.

This requirement applies to the shell environment and each subshell environment separately. For example, in:

```
set −e; (false; echo one) | cat; echo two
```

the *false* command causes the subshell to exit without executing `echo one`; however, `echo two` is executed because the exit status of the pipeline `(false; echo one) | cat` is zero.

In

```
set −e; echo $(false; echo one) two
```

the *false* command causes the subshell in which the command substitution is performed to exit without executing `echo one`; the exit status of the subshell is ignored and the shell then executes the word-expanded command `echo two`.

−f The shell shall disable pathname expansion.

OB **−h** Setting this option may speed up *PATH* searches (see XBD Chapter 8). This option may be enabled by default.

−m This option shall be supported if the implementation supports the User Portability Utilities option. When this option is enabled, the shell shall perform job control actions as described in Section 2.11. This option shall be enabled by default for interactive shells.

−n The shell shall read commands but does not execute them; this can be used to check for shell script syntax errors. Interactive shells and subshells of interactive shells, recursively, may ignore this option.

−o Write the current settings of the options to standard output in an unspecified format.

+o Write the current option settings to standard output in a format that is suitable for reinput to the shell as commands that achieve the same options settings.

−o *option*
Set various options, many of which shall be equivalent to the single option letters. The following values of *option* shall be supported:

allexport Equivalent to **−a**.

errexit Equivalent to –**e**.

ignoreeof Prevent an interactive shell from exiting on end-of-file. This setting prevents accidental logouts when <control>-D is entered. A user shall explicitly *exit* to leave the interactive shell. This option shall be supported if the system supports the User Portability Utilities option.

monitor Equivalent to –**m**. This option shall be supported if the system supports the User Portability Utilities option.

noclobber Equivalent to –**C** (uppercase C).

noglob Equivalent to –**f**.

noexec Equivalent to –**n**.

OB *nolog* Prevent the entry of function definitions into the command history; see Command History List. This option may have no effect; it is kept for compatibility with previous versions of the standard. This option shall be supported if the system supports the User Portability Utilities option.

notify Equivalent to –**b**.

nounset Equivalent to –**u**.

pipefail Derive the exit status of a pipeline from the exit statuses of all of the commands in the pipeline, not just the last (rightmost) command, as described in Section 2.9.2.

verbose Equivalent to –**v**.

vi Allow shell command line editing using the built-in *vi* editor. Enabling *vi* mode shall disable any other command line editing mode provided as an implementation extension. This option shall be supported if the system supports the User Portability Utilities option.

It need not be possible to set *vi* mode on for certain block-mode terminals.

xtrace Equivalent to –**x**.

–**u** When the shell tries to expand, in a parameter expansion or an arithmetic expansion, an unset parameter other than the `'@'` and `'*'` special parameters, it shall write a message to standard error and the expansion shall fail with the consequences specified in Section 2.8.1.

–**v** The shell shall write its input to standard error as it is read.

–**x** The shell shall write to standard error a trace for each command after it expands the command and before it executes it. It is unspecified whether the command that turns tracing off is traced.

The default for all these options shall be off (unset) unless stated otherwise in the description of the option or unless the shell was invoked with them on; see *sh*.

The remaining arguments shall be assigned in order to the positional parameters. The special parameter `'#'` shall be set to reflect the number of positional parameters. All positional parameters shall be unset before any new values are assigned.

If the first argument is `'–'`, the results are unspecified.

The special argument `"––"` immediately following the *set* command name can be used to delimit the arguments if the first argument begins with `'+'` or `'–'`, or to prevent inadvertent listing of all shell variables when there are no arguments. The command *set* –– without *argument*

shall unset all positional parameters and set the special parameter `'#'` to zero.

OPTIONS

See the DESCRIPTION.

OPERANDS

See the DESCRIPTION.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

Default.

STDOUT

See the DESCRIPTION.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

0 Successful completion.

>0 An invalid option was specified, or an error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Application writers should avoid relying on *set* **−e** within functions. For example, in the following script:

```
set −e
start() {
    some_server
    echo some_server started successfully
}
start || echo >&2 some_server failed
```

the **−e** setting is ignored within the function body (because the function is a command in an AND-OR list other than the last). Therefore, if `some_server` fails, the function carries on to echo `"some_server started successfully"`, and the exit status of the function is zero (which means `"some_server failed"` is not output).

Use of *set* **−n** causes the shell to parse the rest of the script without executing any commands, meaning that *set* **+n** cannot be used to undo the effect. Syntax checking is more commonly done via `sh −n` *`script_name`*.

EXAMPLES

Write out all variables and their values:

```
set
```

Set $1, $2, and $3 and set `"$#"` to 3:

```
set c a b
```

Turn on the **−x** and **−v** options:

```
set -xv
```

Unset all positional parameters:

```
set --
```

Set $1 to the value of *x*, even if it begins with `'−'` or `'+'`:

```
set -- "$x"
```

Set the positional parameters to the expansion of *x*, even if *x* expands with a leading `'−'` or `'+'`:

```
set -- $x
```

RATIONALE

The *set* −− form is listed specifically in the SYNOPSIS even though this usage is implied by the Utility Syntax Guidelines. The explanation of this feature removes any ambiguity about whether the *set* −− form might be misinterpreted as being equivalent to *set* without any options or arguments. The functionality of this form has been adopted from the KornShell. In System V, *set* −− only unsets parameters if there is at least one argument; the only way to unset all parameters is to use *shift*. Using the KornShell version should not affect System V scripts because there should be no reason to issue it without arguments deliberately; if it were issued as, for example:

```
set -- "$@"
```

and there were in fact no arguments resulting from `"$@"`, unsetting the parameters would have no result.

The *set* + form in early proposals was omitted as being an unnecessary duplication of *set* alone and not widespread historical practice.

The *noclobber* option was changed to allow *set* **−C** as well as the *set* **−o** *noclobber* option. The single-letter version was added so that the historical `"$-"` paradigm would not be broken; see Section 2.5.2.

The description of the **−e** option is intended to match the behavior of the 1988 version of the KornShell.

The **−h** option is related to command name hashing. See *hash*. The normative description is deliberately vague because the way this option works varies between shell implementations.

Earlier versions of this standard specified **−h** as a way to locate and remember utilities to be invoked by functions as those functions are defined (the utilities are normally located when the function is executed). However, this did not match existing practice in most shells.

The following *set* options were omitted intentionally with the following rationale:

−k The **−k** option was originally added by the author of the Bourne shell to make it easier for users of pre-release versions of the shell. In early versions of the Bourne shell the construct *set name=value* had to be used to assign values to shell variables. The problem with **−k** is that the behavior affects parsing, virtually precluding writing any compilers. To explain the

behavior of **−k**, it is necessary to describe the parsing algorithm, which is implementation-defined. For example:

```
set -k; echo name=value
```

and:

```
set -k
echo name=value
```

behave differently. The interaction with functions is even more complex. What is more, the **−k** option is never needed, since the command line could have been reordered.

−t The **−t** option is hard to specify and almost never used. The only known use could be done with here-documents. Moreover, the behavior with *ksh* and *sh* differs. The reference page says that it exits after reading and executing one command. What is one command? If the input is *date;date*, *sh* executes both *date* commands while *ksh* does only the first.

Consideration was given to rewriting *set* to simplify its confusing syntax. A specific suggestion was that the *unset* utility should be used to unset options instead of using the non-*getopt*( )-able *+option* syntax. However, the conclusion was reached that the historical practice of using *+option* was satisfactory and that there was no compelling reason to modify such widespread historical practice.

The **−o** option was adopted from the KornShell to address user needs. In addition to its generally friendly interface, **−o** is needed to provide the *vi* command line editing mode, for which historical practice yields no single-letter option name. (Although it might have been possible to invent such a letter, it was recognized that other editing modes would be developed and **−o** provides ample name space for describing such extensions.)

Historical implementations are inconsistent in the format used for **−o** option status reporting. The **+o** format without an option-argument was added to allow portable access to the options that can be saved and then later restored using, for instance, a dot script.

Historically, *sh* did trace the command *set* **+x**, but *ksh* did not.

The *ignoreeof* setting prevents accidental logouts when the end-of-file character (typically <control>-D) is entered. A user shall explicitly *exit* to leave the interactive shell.

The *set* **−m** option was added to apply only to the UPE because it applies primarily to interactive use, not shell script applications.

The ability to do asynchronous notification became available in the 1988 version of the KornShell. To have it occur, the user had to issue the command:

```
trap "jobs -n" CLD
```

The C shell provides two different levels of an asynchronous notification capability. The environment variable *notify* is analogous to what is done in *set* **−b** or *set* **−o** *notify*. When set, it notifies the user immediately of background job completions. When unset, this capability is turned off.

The other notification ability comes through the built-in utility *notify*. The syntax is:

```
notify [%job ... ]
```

By issuing *notify* with no operands, it causes the C shell to notify the user asynchronously when the state of the current job changes. If given operands, *notify* asynchronously informs the user of changes in the states of the specified jobs.

To add asynchronous notification to the POSIX shell, neither the KornShell extensions to *trap*,

nor the C shell *notify* environment variable seemed appropriate (*notify* is not a proper POSIX environment variable name).

The *set* −**b** option was selected as a compromise.

The *notify* built-in was considered to have more functionality than was required for simple asynchronous notification.

Historically, some shells applied the −**u** option to all parameters including `$@` and `$*`. The standard developers felt that this was a misfeature since it is normal and common for `$@` and `$*` to be used in shell scripts regardless of whether they were passed any arguments. Treating these uses as an error when no arguments are passed reduces the value of −**u** for its intended purpose of finding spelling mistakes in variable names and uses of unset positional parameters.

FUTURE DIRECTIONS

A future version of this standard may remove the −**o** *nolog* option.

SEE ALSO

Section 2.15, *hash*

XBD Section 4.26, Section 12.2

CHANGE HISTORY

Issue 6

The obsolescent *set* command name followed by `'−'` has been removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The *nolog* option is added to *set* −**o**.

IEEE PASC Interpretation 1003.2 #167 is applied, clarifying that the options default also takes into account the description of the option.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page sections use terms as described in the Utility Description Defaults (Section 1.4). No change in behavior is intended.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/8 is applied, changing the square brackets in the example in RATIONALE to be in bold, which is the typeface used for optional items.

Issue 7

Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first argument is `'−'`.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

XSI shading is removed from the −**h** functionality.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0046 [52], XCU/TC1-2008/0047 [155,280], XCU/TC1-2008/0048 [52], XCU/TC1-2008/0049 [52], and XCU/TC1-2008/0050 [155,430] are applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0053 [584], XCU/TC2-2008/0054 [717], XCU/TC2-2008/0055 [717], and XCU/TC2-2008/0056 [960] are applied.

Issue 8

Austin Group Defect 559 is applied, changing the description of the –**u** option.

Austin Group Defect 789 is applied, adding –**o** *pipefail*.

Austin Group Defect 981 is applied, changing the description of the –**o** *nolog* option and the FUTURE DIRECTIONS section.

Austin Group Defects 1009 and 1555 are applied, changing the description of the –**a** option.

Austin Group Defect 1016 is applied, changing the description of the –**C** option.

Austin Group Defect 1055 is applied, adding a paragraph about the –**n** option to the APPLICATION USAGE section.

Austin Group Defect 1063 is applied, changing the description of the –**h** option.

Austin Group Defect 1150 is applied, changing the description of the –**e** option.

Austin Group Defect 1207 is applied, clarifying which option-arguments of the –**o** option are related to the User Portability Utilities option.

Austin Group Defect 1254 is applied, changing the descriptions of the –**b** and –**m** options.

Austin Group Defect 1384 is applied, allowing subshells of interactive shells to ignore the –**n** option.

NAME

shift — shift positional parameters

SYNOPSIS

```
shift [n]
```

DESCRIPTION

The positional parameters shall be shifted. Positional parameter 1 shall be assigned the value of parameter (1+*n*), parameter 2 shall be assigned the value of parameter (2+*n*), and so on. The parameters represented by the numbers `"$#"` down to `"$#-n+1"` shall be unset, and the parameter `'#'` is updated to reflect the new number of positional parameters.

The value *n* shall be an unsigned decimal integer less than or equal to the value of the special parameter `'#'`. If *n* is not given, it shall be assumed to be 1. If *n* is 0, the positional and special parameters are not changed.

OPTIONS

None.

OPERANDS

See the DESCRIPTION.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages and the warning message specified in EXIT STATUS.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

If the *n* operand is invalid or is greater than `"$#"`, this may be treated as an error and a non-interactive shell may exit; if the shell does not exit in this case, a non-zero exit status shall be returned and a warning message shall be written to standard error. Otherwise, zero shall be returned.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

```
$ set a b c d e
$ shift 2
$ echo $*
c d e
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.15

CHANGE HISTORY

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page sections use terms as described in the Utility Description Defaults (Section 1.4). No change in behavior is intended.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0051 [459] is applied.

Issue 8

Austin Group Defect 1265 is applied, updating the EXIT STATUS and STDERR sections to align with the changes made to Section 2.8.1 between Issue 6 and Issue 7.

NAME

times — write process times

SYNOPSIS

```
times
```

DESCRIPTION

The *times* utility shall write the accumulated user and system times for the shell and for all of its child processes, in the following POSIX locale format:

```
"%dm%fs %dm%fs\n%dm%fs %dm%fs\n", <shell user minutes>,
    <shell user seconds>, <shell system minutes>,
    <shell system seconds>, <children user minutes>,
    <children user seconds>, <children system minutes>,
    <children system seconds>
```

The four pairs of times shall correspond to the members of the **<sys/times.h> tms** structure (defined in XBD Chapter 14) as returned by *times*( ): *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime*, respectively.

OPTIONS

None.

OPERANDS

None.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

Default.

STDOUT

See the DESCRIPTION.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

```
$ times
0m0.43s 0m1.11s
8m44.18s 1m43.23s
```

RATIONALE

The *times* special built-in from the Single UNIX Specification is now required for all conforming shells.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.15

XBD **<sys/times.h>**

CHANGE HISTORY

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/9 is applied, changing text in the DESCRIPTION from: ``Write the accumulated user and system times for the shell and for all of its child processes ...'' to: ``The *times* utility shall write the accumulated user and system times for the shell and for all of its child processes ...''.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0056 [960] is applied.

NAME

trap — trap signals

SYNOPSIS

```
trap n [condition...]
trap -p [condition...]
trap [action condition...]
```

DESCRIPTION

If the **−p** option is not specified and the first operand is an unsigned decimal integer, the shell shall treat all operands as conditions, and shall reset each condition to the default value. Otherwise, if the **−p** option is not specified and there are operands, the first operand shall be treated as an action and the remaining as conditions.

If *action* is `'-'`, the shell shall reset each *condition* to the default value. If *action* is null (`""`), the shell shall ignore each specified *condition* if it arises. Otherwise, the argument *action* shall be read and executed by the shell when one of the corresponding conditions arises. The action of *trap* shall override a previous action (either default action or one explicitly set). The value of `"$?"` after the *trap* action completes shall be the value it had before the *trap* action was executed.

The condition can be EXIT, 0 (equivalent to EXIT), or a signal specified using a symbolic name, without the SIG prefix, as listed in the tables of signal names in the **<signal.h>** header defined in XBD Chapter 14; for example, HUP, INT, QUIT, TERM. Implementations may permit names with the SIG prefix or ignore case in signal names as an extension. Setting a trap for SIGKILL or SIGSTOP produces undefined results.

The EXIT condition shall occur when the shell terminates normally (exits), and may occur when the shell terminates abnormally as a result of delivery of a signal (other than SIGKILL) whose *trap* action is the default.

The environment in which the shell executes a *trap* action on EXIT shall be identical to the environment immediately after the last command executed before the *trap* action on EXIT was executed.

If *action* is neither `'-'` nor the empty string, then each time a matching *condition* arises, the *action* shall be executed in a manner equivalent to:

```
eval action
```

Signals that were ignored on entry to a non-interactive shell cannot be trapped or reset, although no error need be reported when attempting to do so. An interactive shell may reset or catch signals ignored on entry. Traps shall remain in place for a given shell until explicitly changed with another *trap* command.

When a subshell is entered, traps that are not being ignored shall be set to the default actions, except in the case of a command substitution containing only a single *trap* command, when the traps need not be altered. Implementations may check for this case using only lexical analysis; for example, if `` `trap` `` and `$( trap -- )` do not alter the traps in the subshell, cases such as assigning `var=trap` and then using `$($var)` may still alter them. This does not imply that the *trap* command cannot be used within the subshell to set new traps.

The *trap* command with no operands shall write to standard output a list of commands associated with each of a set of conditions; if the **−p** option is not specified, this set shall contain only the conditions that are not in the default state (including signals that were ignored on entry to a non-interactive shell); if the **−p** option is specified, the set shall contain all conditions, except that it is unspecified whether conditions corresponding to the SIGKILL and SIGSTOP signals are included in the set. If the command is executed in a subshell, the implementation does not

perform the optional check described above for a command substitution containing only a single *trap* command, and no *trap* commands with operands have been executed since entry to the subshell, the list shall contain the commands that were associated with each condition immediately before the subshell environment was entered. Otherwise, the list shall contain the commands currently associated with each condition. The format shall be:

```
"trap -- %s %s ...\n", <action>, <condition> ...
```

The shell shall format the output, including the proper use of quoting, so that it is suitable for reinput to the shell as commands that achieve the same trapping results for the set of conditions included in the output, except for signals that were ignored on entry to the shell as described above. If this set includes conditions corresponding to the SIGKILL and SIGSTOP signals, the shell shall accept them when the output is reinput to the shell (where accepting them means they do not cause a non-zero exit status, a diagnostic message, or undefined behavior). For example:

```
save_traps=$(trap -p)
...
eval "$save_traps"
```

or:

```
save_traps=$(trap -p INT QUIT)
trap "some command" INT QUIT
...
eval "$save_traps"
```

XSI XSI-conformant systems also allow numeric signal numbers for the conditions corresponding to the following signal names:

1 SIGHUP

2 SIGINT

3 SIGQUIT

6 SIGABRT

9 SIGKILL

14 SIGALRM

15 SIGTERM

XSI If an invalid signal name or number is specified, the *trap* utility shall write a warning message to standard error.

The *trap* special built-in shall conform to XBD Section 12.2.

OPTIONS

The following option shall be supported:

−p Write to standard output a list of commands associated with each *condition* operand. The behavior when there are no operands is specified in the DESCRIPTION section.

The shell shall format the output, including the proper use of quoting, so that it is suitable for reinput to the shell as commands that achieve the same trapping results for the specified set of conditions. If a *condition* operand is a condition corresponding to the SIGKILL or

SIGSTOP signal, and *trap* **−p** without any operands would not include it in the set of conditions for which it writes output, the behavior is undefined if the output is reinput to the shell.

OPERANDS

See the DESCRIPTION.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

Default.

STDOUT

See the DESCRIPTION.

STDERR

The standard error shall be used only for diagnostic messages and warning messages about
XSI invalid signal names or numbers.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

XSI If the trap name or number is invalid, a non-zero exit status shall be returned; otherwise, zero
XSI shall be returned. For both interactive and non-interactive shells, invalid signal names or numbers shall not be considered an error and shall not cause the shell to abort.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

When the **−p** option is not used, since *trap* with no operands does not output commands to restore traps that are currently set to default, these need to be restored separately. The RATIONALE section shows examples and describes their drawbacks.

EXAMPLES

Write out a list of all traps and actions:

```
trap
```

Set a trap so the *logout* utility in the directory referred to by the *HOME* environment variable executes when the shell terminates:

```
trap '"$HOME"/logout' EXIT
```

or:

```
trap '"$HOME"/logout' 0
```

Unset traps on INT, QUIT, TERM, and EXIT:

```
trap - INT QUIT TERM EXIT
```


RATIONALE

Implementations may permit lowercase signal names as an extension. Implementations may also accept the names with the SIG prefix; no known historical shell does so. The *trap* and *kill* utilities in this volume of POSIX.1-2024 are now consistent in their omission of the SIG prefix for signal names. Some *kill* implementations do not allow the prefix, and *kill* **–l** lists the signals without prefixes.

Trapping SIGKILL or SIGSTOP is syntactically accepted by some historical implementations, but it has no effect. Portable POSIX applications cannot attempt to trap these signals.

The output format is not historical practice. Since the output of historical *trap* commands is not portable (because numeric signal values are not portable) and had to change to become so, an opportunity was taken to format the output in a way that a shell script could use to save and then later reuse a trap if it wanted.

The KornShell uses an **ERR** trap that is triggered whenever *set* **–e** would cause an exit. This is allowable as an extension, but was not mandated, as other shells have not used it.

The text about the environment for the EXIT trap invalidates the behavior of some historical versions of interactive shells which, for example, close the standard input before executing a trap on 0. For example, in some historical interactive shell sessions the following trap on 0 would always print `"--"`:

```
trap 'read foo; echo "-$foo-"' 0
```

The command:

```
trap 'eval " $cmd"' 0
```

causes the contents of the shell variable *cmd* to be executed as a command when the shell exits. Using:

```
trap '$cmd' 0
```

does not work correctly if *cmd* contains any special characters such as quoting or redirections. Using:

```
trap " $cmd" 0
```

also works (the leading **<space>** character protects against unlikely cases where *cmd* is a decimal integer or begins with `'-'`), but it expands the *cmd* variable when the *trap* command is executed, not when the exit action is executed.

The **–p** option was added because without it the method used to restore traps needs to include special handling of traps that are set to default when *trap* with no operands is used to save the current traps. One example is:

```
save_traps=$(trap)
trap "some command" INT QUIT
save_traps="trap - INT QUIT; $save_traps"
...
eval "$save_traps"
```

but this method relies on hard-coding the commands to reset the traps that are being set. It also has a race condition if INT or QUIT was not set to default when saved, since it first sets them to default and then restores the saved traps. A more general approach would be:

```
save_traps=$(trap)
for sig in EXIT $( kill -l )
```



```
    do
        case "$sig" in
        SIGKILL | KILL | sigkill | kill | SIGSTOP | STOP | sigstop | stop)
        ;;
        *) trap - $sig
        ;;
        esac
    done
    eval "$save_traps"
```

This has the same race condition since it first sets all traps (that can be set) to default and then restores those that were not previously set to default.

Historically, some shells behaved the same with and without **−p** when there are no operands. This standard requires that the set of conditions differs between the two cases: with **−p** it is all conditions (except possibly SIGKILL and SIGSTOP); without **−p** it is only the conditions that are not in the default state.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.15

XBD Section 12.2, **<signal.h>**

CHANGE HISTORY

Issue 6

XSI-conforming implementations provide the mapping of signal names to numbers given above (previously this had been marked obsolescent). Other implementations need not provide this optional mapping.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page sections use terms as described in the Utility Description Defaults (Section 1.4). No change in behavior is intended.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Austin Group Interpretation 1003.1-2001 #116 is applied.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0052 [53,268,440], XCU/TC1-2008/0053 [53,268,440], XCU/TC1-2008/0054 [163], XCU/TC1-2008/0055 [163], and XCU/TC1-2008/0056 [163] are applied.

Issue 8

Austin Group Defect 621 is applied, clarifying when the EXIT condition occurs.

Austin Group Defect 1029 is applied, clarifying the execution of *trap* actions.

Austin Group Defects 1211 and 1212 are applied, adding the **−p** option and clarifying that, when **−p** is not specified, the output of *trap* with no operands does not list conditions that are in the default state.

Austin Group Defect 1265 is applied, updating the DESCRIPTION, STDERR and EXIT STATUS sections to align with the changes made to Section 2.8.1 between Issue 6 and Issue 7.

83897 Austin Group Defect 1285 is applied, inserting a blank line between the two SYNOPSIS lines.

NAME

unset — unset values and attributes of variables and functions

SYNOPSIS

```
unset [-fv] name...
```

DESCRIPTION

The *unset* utility shall unset each variable or function definition specified by *name* that does not have the *readonly* attribute and remove any attributes other than *readonly* that have been given to *name* (see Section 2.15 *export* and *readonly*).

If **−v** is specified, *name* refers to a variable name and the shell shall unset it and remove it from the environment. Read-only variables cannot be unset.

If **−f** is specified, *name* refers to a function and the shell shall unset the function definition.

If neither **−f** nor **−v** is specified, *name* refers to a variable; if a variable by that name does not exist, it is unspecified whether a function by that name, if any, shall be unset.

Unsetting a variable or function that was not previously set shall not be considered an error and does not cause the shell to abort.

The *unset* special built-in shall support XBD Section 12.2.

Note that:

```
VARIABLE=
```

is not equivalent to an *unset* of **VARIABLE**; in the example, **VARIABLE** is set to `""`. Also, the variables that can be *unset* should not be misinterpreted to include the special parameters (see Section 2.5.2).

OPTIONS

See the DESCRIPTION.

OPERANDS

See the DESCRIPTION.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

0 All *name* operands were successfully unset.

>0 At least one *name* could not be unset.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

Unset *VISUAL* variable:

```
unset -v VISUAL
```

Unset the functions **foo** and **bar**:

```
unset -f foo bar
```

RATIONALE

Consideration was given to omitting the **−f** option in favor of an *unfunction* utility, but the standard developers decided to retain historical practice.

The **−v** option was introduced because System V historically used one name space for both variables and functions. When *unset* is used without options, System V historically unset either a function or a variable, and there was no confusion about which one was intended. A portable POSIX application can use *unset* without an option to unset a variable, but not a function; the **−f** option must be used.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.15

XBD Section 12.2

CHANGE HISTORY

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page sections use terms as described in the Utility Description Defaults (Section 1.4). No change in behavior is intended.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 1075 is applied, clarifying that *unset* removes attributes, other than *readonly*, from the variables it unsets.

Chapter 3
Utilities

This chapter contains the definitions of the utilities, as follows:

- Mandatory utilities that are present on every conformant system
- Optional utilities that are present only on systems supporting the associated option; see Section 1.8.1 (on page 7) for information on the options in this volume of POSIX.1-2024

NAME

admin — create and administer SCCS files (**DEVELOPMENT**)

SYNOPSIS

XSI

```
admin -i[name] [-n] [-a login] [-d flag] [-e login] [-f flag]
    [-m mrlist] [-r rel] [-t[name] [-y[comment]] newfile

admin -n [-a login] [-d flag] [-e login] [-f flag] [-m mrlist]
    [-t[name]] [-y[comment]] newfile...

admin [-a login] [-d flag] [-m mrlist] [-r rel] [-t[name]] file...

admin -h file...

admin -z file...
```

DESCRIPTION

The *admin* utility shall create new SCCS files or change parameters of existing ones. If a named file does not exist, it shall be created, and its parameters shall be initialized according to the specified options. Parameters not initialized by an option shall be assigned a default value. If a named file does exist, parameters corresponding to specified options shall be changed, and other parameters shall be left as is.

All SCCS filenames supplied by the application shall be of the form s.*filename*. New SCCS files shall be given read-only permission mode. Write permission in the parent directory is required to create a file. All writing done by *admin* shall be to a temporary *x-file*, named x.*filename* (see *get*) created with read-only mode if *admin* is creating a new SCCS file, or created with the same mode as that of the SCCS file if the file already exists. After successful execution of *admin*, the SCCS file shall be removed (if it exists), and the *x-file* shall be renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occur.

The *admin* utility shall also use a transient lock file (named z.*filename*), which is used to prevent simultaneous updates to the SCCS file; see *get*.

OPTIONS

The *admin* utility shall conform to XBD Section 12.2 (on page 215), except that the **–i**, **–t**, and **–y** options have optional option-arguments. These optional option-arguments shall not be presented as separate arguments. The following options are supported:

–n Create a new SCCS file. When **–n** is used without **–i**, the SCCS file shall be created with control information but without any file data.

–i[*name***]** Specify the *name* of a file from which the text for a new SCCS file shall be taken. The text constitutes the first delta of the file (see the **–r** option for the delta numbering scheme). If the **–i** option is used, but the *name* option-argument is omitted, the text shall be obtained by reading the standard input. If this option is omitted, the SCCS file shall be created with control information but without any file data. The **–i** option implies the **–n** option.

–r *SID* Specify the SID of the initial delta to be inserted. This SID shall be a trunk SID; that is, the branch and sequence numbers shall be zero or missing. The level number is optional, and defaults to 1.

–t[*name***]** Specify the *name* of a file from which descriptive text for the SCCS file shall be taken. In the case of existing SCCS files (neither **–i** nor **–n** is specified):

- A **–t** option without a *name* option-argument shall cause the removal of descriptive text (if any) currently in the SCCS file.
- A **–t** option with a *name* option-argument shall cause the text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.

–f *flag* Specify a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. Several **–f** options may be supplied on a single *admin* command line. Implementations shall recognize the following flags and associated values:

b Allow use of the **–b** option on a *get* command to create branch deltas.

c*ceil* Specify the highest release (that is, ceiling), a number less than or equal to 9 999, which may be retrieved by a *get* command for editing. The default value for an unspecified **c** flag shall be 9 999.

f*floor* Specify the lowest release (that is, floor), a number greater than 0 but less than 9 999, which may be retrieved by a *get* command for editing. The default value for an unspecified **f** flag shall be 1.

d*SID* Specify the default delta number (SID) to be used by a *get* command.

i*str* Treat the ``No ID keywords'' message issued by *get* or *delta* as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords (see *get*) are found in the text retrieved or stored in the SCCS file. If a value is supplied, the application shall ensure that the keywords exactly match the given string; however, the string shall contain a keyword, and no embedded <newline> characters.

j Allow concurrent *get* commands for editing on the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.

l*list* Specify a *list* of releases to which deltas can no longer be made (that is, *get* **–e** against one of these locked releases fails). Conforming applications shall use the following syntax to specify a *list*. Implementations may accept additional forms as an extension:

```
<list> ::= a | <range-list>
<range-list> ::= <range> | <range-list>, <range>
<range> ::= <SID>
```

The character *a* in the *list* shall be equivalent to specifying all releases for the named SCCS file. The non-terminal **<***SID***>** in range shall be the delta number of an existing delta associated with the SCCS file.

n Cause *delta* to create a null delta in each of those releases (if any) being skipped when a delta is made in a new release (for example, in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas shall serve as anchor points so that branch deltas may later be created from them. The absence of this flag shall cause skipped releases to be nonexistent in the SCCS file, preventing branch deltas from being created from them in the future. During the initial creation of an SCCS file, the **n** flag may be ignored; that is, if the **–r** option is used to set the release number of the initial SID to a value greater than 1, null deltas need not be created for the ``skipped'' releases.

84067 **q***text* Substitute user-definable *text* for all occurrences of the %**Q**% keyword in
84068 the SCCS file text retrieved by *get*.

84069 **m***mod* Specify the module name of the SCCS file substituted for all occurrences
84070 of the %**M**% keyword in the SCCS file text retrieved by *get*. If the **m** flag
84071 is not specified, the value assigned shall be the name of the SCCS file with
84072 the leading `'.'` removed.

84073 **t***type* Specify the *type* of module in the SCCS file substituted for all occurrences
84074 of the %**Y**% keyword in the SCCS file text retrieved by *get*.

84075 **v***pgm* Cause *delta* to prompt for modification request (MR) numbers as the
84076 reason for creating a delta. The optional value specifies the name of an
84077 MR number validation program. (If this flag is set when creating an SCCS
84078 file, the application shall ensure that the **m** option is also used even if its
84079 value is null.)

84080 −**d** *flag* Remove (delete) the specified *flag* from an SCCS file. Several −**d** options may be
84081 supplied on a single *admin* command. See the −**f** option for allowable *flag* names.
84082 (The **l***list* flag gives a *list* of releases to be unlocked. See the −**f** option for further
84083 description of the **l** flag and the syntax of a *list*.)

84084 −**a** *login* Specify a *login* name, or numerical group ID, to be added to the list of users who
84085 may make deltas (changes) to the SCCS file. A group ID shall be equivalent to
84086 specifying all *login* names common to that group ID. Several −**a** options may be
84087 used on a single *admin* command line. As many *login*s, or numerical group IDs, as
84088 desired may be on the list simultaneously. If the list of users is empty, then anyone
84089 may add deltas. If *login* or group ID is preceded by a `'!'`, the users so specified
84090 shall be denied permission to make deltas.

84091 −**e** *login* Specify a *login* name, or numerical group ID, to be erased from the list of users
84092 allowed to make deltas (changes) to the SCCS file. Specifying a group ID is
84093 equivalent to specifying all *login* names common to that group ID. Several −**e**
84094 options may be used on a single *admin* command line.

84095 −**y**[*comment*] Insert the *comment* text into the SCCS file as a comment for the initial delta in a
84096 manner identical to that of *delta*. In the POSIX locale, omission of the −**y** option
84097 shall result in a default comment line being inserted in the form:

84098 `"date and time created %s %s by %s", <`*date*`>, <`*time*`>, <`*login*`>`

84099 where **<***date***>** is expressed in the format of the *date* utility's `%y/%m/%d` conversion
84100 specification, **<***time***>** in the format of the *date* utility's `%T` conversion specification
84101 format, and **<***login***>** is the login name of the user creating the file.

84102 −**m** *mrlist* Insert the list of modification request (MR) numbers into the SCCS file as the
84103 reason for creating the initial delta in a manner identical to *delta*. The application
84104 shall ensure that the **v** flag is set and the MR numbers are validated if the **v** flag has
84105 a value (the name of an MR number validation program). A diagnostic message
84106 shall be written if the **v** flag is not set or MR validation fails.

84107 −**h** Check the structure of the SCCS file and compare the newly computed checksum
84108 with the checksum that is stored in the SCCS file. If the newly computed checksum
84109 does not match the checksum in the SCCS file, a diagnostic message shall be
84110 written.

84111 **–z** Recompute the SCCS file checksum and store it in the first line of the SCCS file (see
84112 the –**h** option above). Note that use of this option on a truly corrupted file may
84113 prevent future detection of the corruption.

84114 **OPERANDS**

84115 The following operands shall be supported:

84116 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *admin*
84117 utility shall behave as though each file in the directory were specified as a named
84118 file, except that non-SCCS files (last component of the pathname does not begin
84119 with **s.**) and unreadable files shall be silently ignored.

84120 *newfile* A pathname of an SCCS file to be created.

84121 If exactly one *file* or *newfile* operand appears, and it is `'–'`, the standard input shall be read; each
84122 line of the standard input shall be taken to be the name of an SCCS file to be processed. Non-
84123 SCCS files and unreadable files shall be silently ignored.

84124 **STDIN**

84125 The standard input shall be a text file used only if –**i** is specified without an option-argument or
84126 if a *file* or *newfile* operand is specified as `'–'`. If the first character of any standard input line is
84127 <SOH> in the POSIX locale, the results are unspecified.

84128 **INPUT FILES**

84129 The existing SCCS files shall be text files of an unspecified format.

84130 The application shall ensure that the file named by the –**i** option's *name* option-argument shall
84131 be a text file; if the first character of any line in this file is <SOH> in the POSIX locale, the results
84132 are unspecified. If this file contains more than 99 999 lines, the number of lines recorded in the
84133 header for this file shall be 99 999 for this delta.

84134 **ENVIRONMENT VARIABLES**

84135 The following environment variables shall affect the execution of *admin*:

84136 *LANG* Provide a default value for the internationalization variables that are unset or null.
84137 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
84138 variables used to determine the values of locale categories.)

84139 *LC_ALL* If set to a non-empty string value, override the values of all the other
84140 internationalization variables.

84141 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
84142 characters (for example, single-byte as opposed to multi-byte characters in
84143 arguments and input files).

84144 *LC_MESSAGES*

84145 Determine the locale that should be used to affect the format and contents of
84146 diagnostic messages written to standard error and the contents of the default –**y**
84147 comment.

84148 *NLSPATH* Determine the location of messages objects and message catalogs.

84149 **ASYNCHRONOUS EVENTS**

84150 Default.

84151 **STDOUT**

84152 Not used.

84153 **STDERR**

84154 The standard error shall be used only for diagnostic messages.

84155 **OUTPUT FILES**

84156 Any SCCS files created shall be text files of an unspecified format. During processing of a *file*, a
84157 locking *z-file*, as described in *get* (on page 2964), may be created and deleted.

84158 **EXTENDED DESCRIPTION**

84159 None.

84160 **EXIT STATUS**

84161 The following exit values shall be returned:

84162 0 Successful completion.

84163 >0 An error occurred.

84164 **CONSEQUENCES OF ERRORS**

84165 Default.

84166 **APPLICATION USAGE**

84167 It is recommended that directories containing SCCS files be writable by the owner only, and that
84168 SCCS files themselves be read-only. The mode of the directories should allow only the owner to
84169 modify SCCS files contained in the directories. The mode of the SCCS files prevents any
84170 modification at all except by SCCS commands.

84171 **EXAMPLES**

84172 None.

84173 **RATIONALE**

84174 None.

84175 **FUTURE DIRECTIONS**

84176 If this utility is directed to display a pathname that contains any bytes that have the encoded
84177 value of a <newline> character when <newline> is a terminator or separator in the output
84178 format being used, implementations are encouraged to treat this as an error. A future version of
84179 this standard may require implementations to treat this as an error.

84180 If this utility is directed to create a new directory entry that contains any bytes that have the
84181 encoded value of a <newline> character, implementations are encouraged to treat this as an
84182 error. A future version of this standard may require implementations to treat this as an error.

84183 **SEE ALSO**

84184 *delta*, *get*, *prs*, *what*

84185 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

84186 **CHANGE HISTORY**

84187 First released in Issue 2.

84188 **Issue 6**

84189 The normative text is reworded to avoid use of the term ``must'' for application requirements,
84190 and to emphasize the term ``shall'' for implementation requirements.

84191 The grammar is updated.

84192 The Open Group Base Resolution bwg2001-007 is applied, adding new text to the INPUT FILES
84193 section warning that the maximum lines recorded in the file is 99 999.

84194 The Open Group Base Resolution bwg2001-009 is applied, amending the description of the **–h**
84195 option.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to behave as follows:

a. Report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

b. Disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

alias — define or display aliases

SYNOPSIS

```
alias [alias-name[=string]...]
```

DESCRIPTION

The *alias* utility shall create or redefine alias definitions or write the values of existing alias definitions to standard output. An alias definition provides a string value that shall replace a command name when it is encountered. For information on valid string values, and the processing involved, see Section 2.3.1 (on page 2477).

An alias definition shall affect the current shell execution environment and the execution environments of the subshells of the current shell. When used as specified by this volume of POSIX.1-2024, the alias definition shall not affect the parent process of the current shell nor any utility environment invoked by the shell; see Section 2.13 (on page 2522).

OPTIONS

None.

OPERANDS

The following operands shall be supported:

alias-name Write the alias definition to standard output.

alias-name=string
Assign the value of *string* to the alias *alias-name*.

If no operands are given, all alias definitions shall be written to standard output.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *alias*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

84245 **STDOUT**

84246 The format for displaying aliases (when no operands or only *name* operands are specified) shall
84247 be:

84248 `"%s=%s\n", name, value`

84249 The *value* string shall be written with appropriate quoting so that it is suitable for reinput to the
84250 shell. See the description of shell quoting in Section 2.2 (on page 2472).

84251 **STDERR**

84252 The standard error shall be used only for diagnostic messages.

84253 **OUTPUT FILES**

84254 None.

84255 **EXTENDED DESCRIPTION**

84256 None.

84257 **EXIT STATUS**

84258 The following exit values shall be returned:

84259 0 Successful completion.

84260 >0 One of the *name* operands specified did not have an alias definition, or an error occurred.

84261 **CONSEQUENCES OF ERRORS**

84262 Default.

84263 **APPLICATION USAGE**

84264 This utility is required to be intrinsic. See Section 1.7 (on page 2470) for details.

84265 Care should be taken to avoid alias values that end with a character that could be treated as part
84266 of an operator token, as it is unspecified whether the character that follows the alias name in the
84267 input can be used as part of the same token (see Section 2.3.1, on page 2477). For example, with:

```
84268 $ alias foo='echo 0'
84269 $ foo>&2
```

84270 the shell can either pass the argument `'0'` to *echo* and redirect fd 1 to fd 2, or pass no arguments
84271 to *echo* and redirect fd 0 to fd 2. Changing it to:

```
84272 $ alias foo='echo "0"'
```

84273 avoids this problem. The alternative of adding a <space> after the `'0'` would also avoid the
84274 problem, but in addition it would alter the way the alias works, as described in Section 2.3.1 (on
84275 page 2477).

84276 Likewise, given:

```
84277 $ alias foo='some_command &'
84278 $ foo&
```

84279 the shell may combine the two `'&'` characters into an `&&` (and) operator. Since the alias cannot
84280 pass arguments to *some_command* and thus can be expected to be invoked without arguments,
84281 adding a <space> after the `'&'` would be an acceptable way to prevent this. Alternatively, the
84282 alias could be specified as a grouping command:

```
84283 $ alias foo='{ some_command & }'
```

84284 Problems can occur for tokens other than operators as well, if the alias is used in unusual ways.
84285 For example, with:


```
$ alias foo='echo $'
$ foo((123))
```

some shells combine the `'$'` and the `"((123))"` to form an arithmetic expansion, but others do not (resulting in a syntax error).

EXAMPLES

1. Create a short alias for a commonly used *ls* command:

   ```
   alias lf="ls -CF"
   ```

2. Create a simple ``redo'' command to repeat previous entries in the command history file:

   ```
   alias r='fc -s'
   ```

3. Use 1K units for *du*:

   ```
   alias du=du\ -k
   ```

4. Set up *nohup* so that it can deal with an argument that is itself an alias name:

   ```
   alias nohup="nohup "
   ```

5. Add the **–F** option to interactive uses of *ls*, even when executed as `xargs ls` or `xargs -0 ls`:

   ```
   alias ls='ls -F'
   alias xargs='xargs '
   alias -- -0='-0 '
   find . [...] -print | xargs ls      # breaks on filenames with \n
                                       # (two aliases expanded)
   find . [...] -print0 | xargs -0 ls  # minimizes \n issues (three
                                       # aliases expanded)
   ```

RATIONALE

The *alias* description is based on historical KornShell implementations. Known differences exist between that and the C shell. The KornShell version was adopted to be consistent with all the other KornShell features in this volume of POSIX.1-2024, such as command line editing.

Since *alias* affects the current shell execution environment, it is generally provided as a shell regular built-in.

Historical versions of the KornShell have allowed aliases to be exported to scripts that are invoked by the same shell. This is triggered by the *alias* **–x** flag; it is allowed by this volume of POSIX.1-2024 only when an explicit extension such as **–x** is used. The standard developers considered that aliases were of use primarily to interactive users and that they should normally not affect shell scripts called by those users; functions are available to such scripts.

Historical versions of the KornShell had not written aliases in a quoted manner suitable for reentry to the shell, but this volume of POSIX.1-2024 has made this a requirement for all similar output. Therefore, consistency was chosen over this detail of historical practice.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.9.5 (on page 2511)

XBD Chapter 8 (on page 167)

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The APPLICATION USAGE section is added.

Issue 7

The *alias* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The first example is changed to remove the creation of an alias for a standard utility that alters its behavior to be non-conforming.

Issue 8

Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that this utility is required to be intrinsic.

Austin Group Defect 953 is applied, clarifying that the details of how alias replacement is performed are in the cross-referenced section (Section 2.3.1, on page 2477) and updating the APPLICATION USAGE section.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1630 is applied, adding a new item in EXAMPLES.

NAME

ar — create and maintain library archives

SYNOPSIS

SD
```
ar -d [-v] archive file...
```

XSI
```
ar -m [-v] archive file...
ar -m -a [-v] posname archive file...
ar -m -b [-v] posname archive file...
ar -m -i [-v] posname archive file...
```

XSI
```
ar -p [-v] [-s] archive [file...]
```

XSI
```
ar -q [-cv] archive file...
```

```
ar -r [-cuv] archive file...
```

XSI
```
ar -r -a [-cuv] posname archive file...
ar -r -b [-cuv] posname archive file...
ar -r -i [-cuv] posname archive file...
```

XSI
```
ar -t [-v] [-s] archive [file...]
```

XSI
```
ar -x [-v] [-sCT] archive [file...]
```

DESCRIPTION

The *ar* utility is part of the Software Development Utilities option.

The *ar* utility can be used to create and maintain groups of files combined into an archive. Once an archive has been created, new files can be added, and existing files in an archive can be extracted, deleted, or replaced. When an archive consists entirely of valid object files, the implementation shall format the archive so that it is usable as a library for link editing (see *c17*). When some of the archived files are not valid object files, the suitability of the archive for library use is undefined. XSI If an archive consists entirely of printable files, the entire archive shall be printable.

When *ar* creates an archive, it creates administrative information indicating whether a symbol table is present in the archive. When there is at least one object file that *ar* recognizes as such in the archive, an archive symbol table shall be created in the archive and maintained by *ar*; it is used by the link editor to search the archive. Whenever the *ar* utility is used to create or update the contents of such an archive, the symbol table shall be rebuilt. The **−s** option shall force the symbol table to be rebuilt.

All *file* operands can be pathnames. However, files within archives shall be named by a filename, which is the last component of the pathname used when the file was entered into the archive. The comparison of *file* operands to the names of files in archives shall be performed by comparing the last component of the operand to the name of the file in the archive.

It is unspecified whether multiple files in the archive may be identically named. In the case of such files, however, each *file* XSI and *posname* operand shall match only the first file in the archive having a name that is the same as the last component of the operand.

OPTIONS

The *ar* utility shall conform to XBD Section 12.2 (on page 215), except for Guideline 9.

The following options shall be supported:

XSI **−a** Position new files in the archive after the file named by the *posname* operand.

XSI **−b** Position new files in the archive before the file named by the *posname* operand.

−c Suppress the diagnostic message that is written to standard error by default when the archive *archive* is created.

XSI **−C** Prevent extracted files from replacing like-named files in the file system. This option is useful when **−T** is also used, to prevent truncated filenames from replacing files with the same prefix.

−d Delete one or more *file*s from *archive*.

XSI **−i** Position new files in the archive before the file in the archive named by the *posname* operand (equivalent to **−b**).

XSI **−m** Move the named files in the archive. The **−a**, **−b**, or **−i** options with the *posname* operand indicate the position; otherwise, move the names files in the archive to the end of the archive.

−p Write the contents of the *file*s in the archive named by *file* operands from *archive* to the standard output. If no *file* operands are specified, the contents of all files in the archive shall be written in the order of the archive.

XSI **−q** Append the named files to the end of the archive. In this case *ar* does not check whether the added files are already in the archive. This is useful to bypass the searching otherwise done when creating a large archive piece by piece.

−r Replace or add *file*s to *archive*. If the archive named by *archive* does not exist, a new archive shall be created and a diagnostic message shall be written to standard error (unless the **−c** option is specified). If no *file*s are specified and the *archive* exists, the results are undefined. Files that replace existing files in the archive shall not change the order of the archive. Files that do not replace existing files in the archive shall be appended to the archive XSI unless a **−a**, **−b**, or **−i** option specifies another position.

XSI **−s** Force the regeneration of the archive symbol table even if *ar* is not invoked with an option that modifies the archive contents. This option is useful to restore the archive symbol table after it has been stripped; see *strip*.

−t Write a table of contents of *archive* to the standard output. Only the files specified by the *file* operands shall be included in the written list. If no *file* operands are specified, all files in *archive* shall be included in the order of the archive.

XSI **−T** Allow filename truncation of extracted files whose archive names are longer than the file system can support. By default, extracting a file with a name that is too long shall be an error; a diagnostic message shall be written and the file shall not be extracted.

−u Update older files in the archive. When used with the **−r** option, files in the archive shall be replaced only if the corresponding *file* has a modification time that is at least as new as the modification time of the file in the archive.

84425 **−v** Give verbose output. When used with the option characters **−d**, **−r**, or **−x**, write a
84426 detailed file-by-file description of the archive creation and maintenance activity, as
84427 described in the STDOUT section.

84428 When used with **−p**, write the name of the file in the archive to the standard output
84429 before writing the file in the archive itself to the standard output, as described in
84430 the STDOUT section.

84431 When used with **−t**, include a long listing of information about the files in the
84432 archive, as described in the STDOUT section.

84433 **−x** Extract the files in the archive named by the *file* operands from *archive*. The
84434 contents of the archive shall not be changed. If no *file* operands are given, all files
84435 in the archive shall be extracted. The modification time of each file extracted shall
84436 be set to the time the file is extracted from the archive.

84437 **OPERANDS**

84438 The following operands shall be supported:

84439 *archive* A pathname of the archive.

84440 *file* A pathname. Only the last component shall be used when comparing against the
84441 names of files in the archive. If two or more *file* operands have the same last
84442 pathname component (basename), the results are unspecified. The
84443 implementation's archive format shall not truncate valid filenames of files added
84444 to or replaced in the archive.

84445 XSI *posname* The name of a file in the archive, used for relative positioning; see options **−m** and
84446 **−r**.

84447 **STDIN**

84448 Not used.

84449 **INPUT FILES**

84450 The archive named by *archive* shall be a file in the format created by *ar* **−r**.

84451 **ENVIRONMENT VARIABLES**

84452 The following environment variables shall affect the execution of *ar*:

84453 *LANG* Provide a default value for the internationalization variables that are unset or null.
84454 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
84455 variables used to determine the values of locale categories.)

84456 *LC_ALL* If set to a non-empty string value, override the values of all the other
84457 internationalization variables.

84458 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
84459 characters (for example, single-byte as opposed to multi-byte characters in
84460 arguments and input files).

84461 *LC_MESSAGES*
84462 Determine the locale that should be used to affect the format and contents of
84463 diagnostic messages written to standard error.

84464 *LC_TIME* Determine the format and content for date and time strings written by *ar* **−tv**.

84465 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

TMPDIR Determine the pathname that overrides the default directory for temporary files, if any.

TZ Determine the timezone used to calculate date and time strings written by *ar* –**tv**. If *TZ* is unset or null, an unspecified default timezone shall be used.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If the –**d** option is used with the –**v** option, the standard output format shall be:

```
"d - %s\n", <file>
```

where *file* is the operand specified on the command line.

If the –**p** option is used with the –**v** option, *ar* shall precede the contents of each file with:

```
"\n<%s>\n\n", <file>
```

where *file* is the operand specified on the command line, if *file* operands were specified, and the name of the file in the archive if they were not.

If the –**r** option is used with the –**v** option:

- If *file* is already in the archive, the standard output format shall be:

  ```
  "r - %s\n", <file>
  ```

  where **<***file***>** is the operand specified on the command line.

- If *file* is not already in the archive, the standard output format shall be:

  ```
  "a - %s\n", <file>
  ```

  where **<***file***>** is the operand specified on the command line.

If the –**t** option is used, *ar* shall write the names of the files in the archive to the standard output in the format:

```
"%s\n", <file>
```

where *file* is the operand specified on the command line, if *file* operands were specified, or the name of the file in the archive if they were not.

If the –**t** option is used with the –**v** option, the standard output format shall be:

```
"%s %u/%u %u %s %d %d:%d %d %s\n", <member mode>, <user ID>,
    <group ID>, <number of bytes in member>,
    <abbreviated month>, <day-of-month>, <hour>,
    <minute>, <year>, <file>
```

where:

<*file***>** Shall be the operand specified on the command line, if *file* operands were specified, or the name of the file in the archive if they were not.

<*member mode***>**
Shall be formatted the same as the **<***file mode***>** string defined in the STDOUT section of *ls*, except that the first character, the **<***entry type***>**, is not used; the string represents the file mode of the file in the archive at the time it was added to or replaced in the archive.

The following represent the last-modification time of a file when it was most recently added to or replaced in the archive:

<*abbreviated month***>**
: Equivalent to the format of the `%b` conversion specification format in *date*.

<*day-of-month***>**
: Equivalent to the format of the `%e` conversion specification format in *date*.

<*hour***>** Equivalent to the format of the `%H` conversion specification format in *date*.

<*minute***>** Equivalent to the format of the `%M` conversion specification format in *date*.

<*year***>** Equivalent to the format of the `%Y` conversion specification format in *date*.

When *LC_TIME* does not specify the POSIX locale, a different format and order of presentation of these fields relative to each other may be used in a format appropriate in the specified locale.

If the **–x** option is used with the **–v** option, the standard output format shall be:

`"x - %s\n", <`*file*`>`

where *file* is the operand specified on the command line, if *file* operands were specified, or the name of the file in the archive if they were not.

STDERR

The standard error shall be used only for diagnostic messages. The diagnostic message about creating a new archive when **–c** is not specified shall not modify the exit status.

OUTPUT FILES

Archives are files with unspecified formats.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

The archive format is not described. It is recognized that there are several known *ar* formats, which are not compatible. The *ar* utility is included, however, to allow creation of archives that are intended for use only on one machine. The archive is specified as a file, and it can be moved as a file. This does allow an archive to be moved from one machine to another machine that uses the same implementation of *ar*.

Utilities such as *pax* (and its forebears *tar* and *cpio*) also provide portable ``archives''. This is a not a duplication; the *ar* utility is included to provide an interface primarily for *make* and the compilers, based on a historical model.

In historical implementations, the –**q** option (available on XSI-conforming systems) is known to execute quickly because *ar* does not check on whether the added members are already in the archive. This is useful to bypass the searching otherwise done when creating a large archive piece-by-piece. These remarks may but need not remain true for a brand new implementation of this utility; hence, these remarks have been moved into the RATIONALE.

BSD implementations historically required applications to provide the –**s** option whenever the archive was supposed to contain a symbol table. As in this volume of POSIX.1-2024, System V historically creates or updates an archive symbol table whenever an object file is removed from, added to, or updated in the archive.

The OPERANDS section requires what might seem to be true without specifying it: the archive cannot truncate the filenames below {NAME_MAX}. Some historical implementations do so, however, causing unexpected results for the application. Therefore, this volume of POSIX.1-2024 makes the requirement explicit to avoid misunderstandings.

According to the System V documentation, the options –**dmpqrtx** are not required to begin with a <hyphen-minus> (`'-'`). This volume of POSIX.1-2024 requires that a conforming application use the leading <hyphen-minus>.

The archive format used by the 4.4 BSD implementation is documented in this RATIONALE as an example:

> A file created by *ar* begins with the ``magic'' string `"!<arch>\n"`. The rest of the archive is made up of objects, each of which is composed of a header for a file, a possible filename, and the file contents. The header is portable between machine architectures, and, if the file contents are printable, the archive is itself printable.
>
> The header is made up of six ASCII fields, followed by a two-character trailer. The fields are the object name (16 characters), the file last modification time (12 characters), the user and group IDs (each 6 characters), the file mode (8 characters), and the file size (10 characters). All numeric fields are in decimal, except for the file mode, which is in octal.
>
> The modification time is the file *st_mtime* field. The user and group IDs are the file *st_uid* and *st_gid* fields. The file mode is the file *st_mode* field. The file size is the file *st_size* field. The two-byte trailer is the string `"`<newline>"`.
>
> Only the name field has any provision for overflow. If any filename is more than 16 characters in length or contains an embedded space, the string `"#1/"` followed by the ASCII length of the name is written in the name field. The file size (stored in the archive header) is incremented by the length of the name. The name is then written immediately following the archive header.
>
> Any unused characters in any of these fields are written as <space> characters. If any fields are their particular maximum number of characters in length, there is no separation between the fields.
>
> Objects in the archive are always an even number of bytes long; files that are an odd number of bytes long are padded with a <newline>, although the size in the header does not reflect this.

The *ar* utility description requires that (when all its members are valid object files) *ar* produce an object code library, which the linkage editor can use to extract object modules. If the linkage editor needs a symbol table to permit random access to the archive, *ar* must provide it; however, *ar* does not require a symbol table.

The BSD –**o** option was omitted. It is a rare conforming application that uses *ar* to extract object

code from a library with concern for its modification time, since this can only be of importance to *make*. Hence, since this functionality is not deemed important for applications portability, the modification time of the extracted files is set to the current time.

There is at least one known implementation (for a small computer) that can accommodate only object files for that system, disallowing mixed object and other files. The ability to handle any type of file is not only historical practice for most implementations, but is also a reasonable expectation.

Consideration was given to changing the output format of *ar* **–tv** to the same format as the output of *ls* **–l**. This would have made parsing the output of *ar* the same as that of *ls*. This was rejected in part because the current *ar* format is commonly used and changes would break historical usage. Second, *ar* gives the user ID and group ID in numeric format separated by a <slash>. Changing this to be the user name and group name would not be correct if the archive were moved to a machine that contained a different user database. Since *ar* cannot know whether the archive was generated on the same machine, it cannot tell what to report.

The text on the **–ur** option combination is historical practice—since one filename can easily represent two different files (for example, **/a/foo** and **/b/foo**), it is reasonable to replace the file in the archive even when the modification time in the archive is identical to that in the file system.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

c17, *date*, *pax*, *strip*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215), **<unistd.h>**, description of {POSIX_NO_TRUNC}

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

This utility is marked as part of the Software Development Utilities option.

The STDOUT description is changed for the **–v** option to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

IEEE PASC Interpretation 1003.2 #198 is applied, changing the description to consistently use ``file'' to refer to a file in the file system hierarchy, ``archive'' to refer to the archive being operated upon by the *ar* utility, and ``file in the archive'' to refer to a copy of a file that is contained in the archive.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/10 is applied, making corrections to the

SYNOPSIS. The change was needed since the **−a**, **−b**, and **−i** options are mutually-exclusive, and *posname* is required if any of these options is specified.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/11 is applied, correcting the description of the two-byte trailer in RATIONALE which had missed out a backquote. The correct trailer is a backquote followed by a <newline>.

Issue 7

SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The description of the **−t** option is changed to say ``Only the files specified ...''.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0057 [584] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to behave as follows:

a. Report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

b. Disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

NAME

asa — interpret carriage-control characters

SYNOPSIS

FR
```
asa [file...]
```

DESCRIPTION

The *asa* utility shall write its input files to standard output, mapping carriage-control characters from the text files to line-printer control sequences in an implementation-defined manner.

The first character of every line shall be removed from the input, and the following actions are performed.

If the character removed is:

<space> The rest of the line is output without change.

0 A <newline> is output, then the rest of the input line.

1 One or more implementation-defined characters that causes an advance to the next page shall be output, followed by the rest of the input line.

+ The <newline> of the previous line shall be replaced with one or more implementation-defined characters that causes printing to return to column position 1, followed by the rest of the input line. If the '+' is the first character in the input, it shall be equivalent to <space>.

The action of the *asa* utility is unspecified upon encountering any character other than those listed above as the first character in a line.

OPTIONS

None.

OPERANDS

file A pathname of a text file used for input. If no *file* operands are specified, the standard input shall be used.

STDIN

The standard input shall be used if no *file* operands are specified, and shall be used if a *file* operand is '−' and the implementation treats the '−' as meaning standard input. Otherwise, the standard input shall not be used. See the INPUT FILES section.

INPUT FILES

The input files shall be text files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *asa*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS
Default.

STDOUT
The standard output shall be the text from the input file modified as described in the DESCRIPTION section.

STDERR
None.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 All input files were output successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
None.

EXAMPLES

1. The following command:

   ```
   asa file
   ```

   permits the viewing of *file* (created by a program using FORTRAN-style carriage-control characters) on a terminal.

2. The following command:

   ```
   a.out | asa | lp
   ```

   formats the FORTRAN output of **a.out** and directs it to the printer.

RATIONALE
The *asa* utility is needed to map ``standard'' FORTRAN 77 output into a form acceptable to contemporary printers. Usually, *asa* is used to pipe data to the *lp* utility; see *lp*.

This utility is generally used only by FORTRAN programs. The standard developers decided to retain *asa* to avoid breaking the historical large base of FORTRAN applications that put carriage-control characters in their output files. There is no requirement that a system have a FORTRAN compiler in order to run applications that need *asa*.

Historical implementations have used an ASCII <form-feed> in response to a 1 and an ASCII <carriage-return> in response to a `'+'`. It is suggested that implementations treat characters other than 0, 1, and `'+'` as <space> in the absence of any compelling reason to do otherwise.

However, the action is listed here as ``unspecified'', permitting an implementation to provide extensions to access fast multiple-line slewing and channel seeking in a non-portable manner.

FUTURE DIRECTIONS

None.

SEE ALSO

lp

XBD Chapter 8 (on page 167)

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is marked as part of the FORTRAN Runtime Utilities option.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #092 is applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

at — execute commands at a later time

SYNOPSIS

```
at [-m] [-f file] [-q queuename] -t time_arg
at [-m] [-f file] [-q queuename] timespec...
at -r at_job_id...
at -l -q queuename
at -l [at_job_id...]
```

DESCRIPTION

The *at* utility shall read commands from standard input and group them together as an *at-job*, to be executed at a later time.

The at-job shall be executed in a separate invocation of the shell, running in a separate process group with no controlling terminal, except that the environment variables, current working directory, file creation mask, and other implementation-defined execution-time attributes in effect when the *at* utility is executed shall be retained and used when the at-job is executed.

When the at-job is submitted, the *at_job_id* and scheduled time shall be written to standard error. The *at_job_id* is an identifier that shall be a string consisting solely of alphanumeric characters and the <period> character. The *at_job_id* shall be assigned by the system when the job is scheduled such that it uniquely identifies a particular job.

User notification and the processing of the job's standard output and standard error are described under the –**m** option.

XSI Users shall be permitted to use *at* if their name appears in the file **at.allow** which is located in an implementation-defined directory. If that file does not exist, the file **at.deny**, which is located in an implementation-defined directory, shall be checked to determine whether the user shall be denied access to *at*. If neither file exists, only a process with appropriate privileges shall be allowed to submit a job. If only **at.deny** exists and is empty, global usage shall be permitted. The **at.allow** and **at.deny** files shall consist of one user name per line.

OPTIONS

The *at* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–**f** *file* Specify the pathname of a file to be used as the source of the at-job, instead of standard input.

–**l** (The letter ell.) Report all jobs scheduled for the invoking user if no *at_job_id* operands are specified. If *at_job_id*s are specified, report only information for these jobs. The output shall be written to standard output.

–**m** Send mail to the invoking user after the at-job has run, announcing its completion. Standard output and standard error produced by the at-job shall be mailed to the user as well, unless redirected elsewhere. Mail shall be sent even if the job produces no output.

If –**m** is not used, the job's standard output and standard error shall be provided to the user by means of mail, unless they are redirected elsewhere; if there is no such output to provide, the implementation need not notify the user of the job's completion.

84797 −**q** *queuename*

84798 Specify in which queue to schedule a job for submission. When used with the −**l**
84799 option, limit the search to that particular queue. By default, at-jobs shall be
84800 scheduled in queue *a*. In contrast, queue *b* shall be reserved for batch jobs; see
84801 *batch*. The meanings of all other *queuename*s are implementation-defined. If −**q** *b* is
84802 specified along with either of the −**t** *time_arg* or *timespec* arguments, the results are
84803 unspecified.

84804 −**r** Remove the jobs with the specified *at_job_id* operands that were previously
84805 scheduled by the *at* utility.

84806 −**t** *time_arg* Submit the job to be run at the time specified by the *time* option-argument, which
84807 the application shall ensure has the format as specified by the *touch* −**t** *time* utility.

84808 **OPERANDS**

84809 The following operands shall be supported:

84810 *at_job_id* The name reported by a previous invocation of the *at* utility at the time the job was
84811 scheduled.

84812 *timespec* Submit the job to be run at the date and time specified. All of the *timespec* operands
84813 are interpreted as if they were separated by **<space>** characters and concatenated,
84814 and shall be parsed as described in the grammar at the end of this section. The date
84815 and time shall be interpreted as being in the timezone of the user (as determined
84816 by the *TZ* variable), unless a timezone name appears as part of *time*, below.

84817 In the POSIX locale, the following describes the three parts of the time specification
84818 string. All of the values from the *LC_TIME* categories in the POSIX locale shall be
84819 recognized in a case-insensitive manner.

84820 *time* The time can be specified as one, two, or four digits. One-digit and
84821 two-digit numbers shall be taken to be hours; four-digit numbers to
84822 be hours and minutes. The time can alternatively be specified as two
84823 numbers separated by a **<colon>**, meaning *hour:minute*. If the
84824 *LC_TIME* category of the locale supports 12-hour time format (see
84825 XBD Section 7.3.5, on page 152), an AM/PM indication in the form of
84826 one of the values from the **am_pm** keywords in the *LC_TIME* locale
84827 category can follow the time; otherwise, a 24-hour clock time shall be
84828 understood. A timezone name can also follow to further qualify the
84829 time. The acceptable timezone names are implementation-defined,
84830 except that they shall be case-insensitive and the string **utc** is
84831 supported to indicate the time is in Coordinated Universal Time. In
84832 the POSIX locale, the *time* field can also be one of the following
84833 tokens:

84834 **midnight** Indicates the time 12:00 am (00:00).

84835 **noon** Indicates the time 12:00 pm.

84836 **now** Indicates the current day and time. Invoking *at* **<now>**
84837 shall submit an at-job for potentially immediate
84838 execution (that is, subject only to unspecified
84839 scheduling delays).

84840 *date* An optional *date* can be specified as either a month name (one of the
84841 values from the **mon** or **abmon** keywords in the *LC_TIME* locale
84842 category) followed by a day number (and possibly year number

preceded by a comma), or a day of the week (one of the values from the **day** or **abday** keywords in the *LC_TIME* locale category). In the POSIX locale, two special days shall be recognized:

today Indicates the current day.

tomorrow Indicates the day following the current day.

If no *date* is given, **today** shall be assumed if the given time is greater than the current time, and **tomorrow** shall be assumed if it is less. If the given month is less than the current month (and no year is given), next year shall be assumed.

increment The optional *increment* shall be a number preceded by a <plus-sign> (`'+'`) and suffixed by one of the following: **minutes**, **hours**, **days**, **weeks**, **months**, or **years**. (The singular forms shall also be accepted.) The keyword **next** shall be equivalent to an increment number of +1. For example, the following are equivalent commands:

```
at 2pm + 1 week
at 2pm next week
```

The following grammar describes the precise format of *timespec* in the POSIX locale. The general conventions for this style of grammar are described in Section 1.3 (on page 2461). This formal syntax shall take precedence over the preceding text syntax description. The longest possible token or delimiter shall be recognized at a given point. When used in a *timespec*, white space shall also delimit tokens.

```
%token hr24clock_hr_min
%token hr24clock_hour
/*
  An hr24clock_hr_min is a one, two, or four-digit number. A one-digit
  or two-digit number constitutes an hr24clock_hour. An hr24clock_hour
  may be any of the single digits [0,9], or may be double digits, ranging
  from [00,23]. If an hr24clock_hr_min is a four-digit number, the
  first two digits shall be a valid hr24clock_hour, while the last two
  represent the number of minutes, from [00,59].
*/

%token wallclock_hr_min
%token wallclock_hour
/*
  A wallclock_hr_min is a one, two-digit, or four-digit number.
  A one-digit or two-digit number constitutes a wallclock_hour.
  A wallclock_hour may be any of the single digits [1,9], or may
  be double digits, ranging from [01,12]. If a wallclock_hr_min
  is a four-digit number, the first two digits shall be a valid
  wallclock_hour, while the last two represent the number of
  minutes, from [00,59].
*/

%token minute
/*
  A minute is a one or two-digit number whose value can be [0,9]
  or [00,59].
*/
```



```
%token day_number
/*
  A day_number is a number in the range appropriate for the particular
  month and year specified by month_name and year_number, respectively.
  If no year_number is given, the current year is assumed if the given
  date and time are later this year. If no year_number is given and
  the date and time have already occurred this year and the month is
  not the current month, next year is the assumed year.
*/

%token year_number
/*
  A year_number is a four-digit number representing the year A.D., in
  which the at_job is to be run.
*/

%token inc_number
/*
  The inc_number is the number of times the succeeding increment
  period is to be added to the specified date and time.
*/

%token timezone_name
/*
  The name of an optional timezone suffix to the time field, in an
  implementation-defined format.
*/

%token month_name
/*
  One of the values from the mon or abmon keywords in the LC_TIME
  locale category.
*/

%token day_of_week
/*
  One of the values from the day or abday keywords in the LC_TIME
  locale category.
*/

%token am_pm
/*
  One of the values from the am_pm keyword in the LC_TIME locale
  category.
*/

%start timespec
%%
timespec    : time
            | time date
            | time increment
            | time date increment
            | nowspec
            ;

nowspec     : "now"
```



```
                    | "now" increment
                    ;

        time        : hr24clock_hr_min
                    | hr24clock_hr_min timezone_name
                    | hr24clock_hour ":" minute
                    | hr24clock_hour ":" minute timezone_name
                    | wallclock_hr_min am_pm
                    | wallclock_hr_min am_pm timezone_name
                    | wallclock_hour ":" minute am_pm
                    | wallclock_hour ":" minute am_pm timezone_name
                    | "noon"
                    | "midnight"
                    ;

        date        : month_name day_number
                    | month_name day_number "," year_number
                    | day_of_week
                    | "today"
                    | "tomorrow"
                    ;

        increment   : "+" inc_number inc_period
                    | "next" inc_period
                    ;

        inc_period  : "minute" | "minutes"
                    | "hour" | "hours"
                    | "day" | "days"
                    | "week" | "weeks"
                    | "month" | "months"
                    | "year" | "years"
                    ;
```

STDIN

The standard input shall be a text file consisting of commands acceptable to the shell command language described in Chapter 2 (on page 2472). The standard input shall only be used if no **–f** *file* option is specified.

INPUT FILES

See the STDIN section.

XSI The text files **at.allow** and **at.deny**, which are located in an implementation-defined directory, shall contain zero or more user names, one per line, of users who are, respectively, authorized or denied access to the *at* and *batch* utilities.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *at*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

84983 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
84984 characters (for example, single-byte as opposed to multi-byte characters in
84985 arguments and input files).

84986 *LC_MESSAGES*
84987 Determine the locale that should be used to affect the format and contents of
84988 diagnostic messages written to standard error and informative messages written to
84989 standard output.

84990 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

84991 *LC_TIME* Determine the format and contents for date and time strings written and accepted
84992 by *at*.

84993 *SHELL* Determine a name of a command interpreter to be used to invoke the at-job. If the
84994 variable is unset or null, *sh* shall be used. If it is set to a value other than a name for
84995 *sh*, the implementation shall do one of the following: use that shell; use *sh*; use the
84996 login shell from the user database; or any of the preceding accompanied by a
84997 warning diagnostic about which was chosen.

84998 *TZ* Determine the timezone. The job shall be submitted for execution at the time
84999 specified by *timespec* or **–t** *time* relative to the timezone specified by the *TZ*
85000 variable. If *timespec* specifies a timezone, it shall override *TZ*. If *timespec* does not
85001 specify a timezone and *TZ* is unset or null, an unspecified default timezone shall
85002 be used.

85003 **ASYNCHRONOUS EVENTS**
85004 Default.

85005 **STDOUT**
85006 When standard input is a terminal, prompts of unspecified format for each line of the user input
85007 described in the STDIN section may be written to standard output.

85008 In the POSIX locale, the following shall be written to the standard output for each job when jobs
85009 are listed in response to the **–l** option:

85010 `"%s\t%s\n", at_job_id, <date>`

85011 where *date* shall be equivalent in format to the output of:

85012 `date +"%a %b %e %T %Y"`

85013 The date and time written shall be adjusted so that they appear in the timezone of the user (as
85014 determined by the *TZ* variable).

85015 **STDERR**
85016 In the POSIX locale, the following shall be written to standard error when a job has been
85017 successfully submitted:

85018 `"job %s at %s\n", at_job_id, <date>`

85019 where *date* has the same format as that described in the STDOUT section. Neither this, nor
85020 warning messages concerning the selection of the command interpreter, shall be considered a
85021 diagnostic that changes the exit status.

85022 Diagnostic messages, if any, shall be written to standard error.

85023 **OUTPUT FILES**

85024 None.

85025 **EXTENDED DESCRIPTION**

85026 None.

85027 **EXIT STATUS**

85028 The following exit values shall be returned:

85029 0 Neither the **–l** option nor the **–r** option was specified and a job was successfully submitted;
85030 or, the **–l** option was specified with no *at_job_id* operands and there were no jobs to be
85031 listed; or, the **–l** option was specified and all job listings were successfully output; or, the **–r**
85032 option was specified and all of the specified jobs were successfully removed.

85033 >0 An error occurred.

85034 **CONSEQUENCES OF ERRORS**

85035 If neither the **–l** option nor the **–r** option was specified, the job shall not be scheduled.
85036 Otherwise, the default actions specified in Section 1.4 (on page 2462) apply.

85037 **APPLICATION USAGE**

85038 The format of the *at* command line shown here is guaranteed only for the POSIX locale. Other
85039 cultures may be supported with substantially different interfaces, although implementations are
85040 encouraged to provide comparable levels of functionality.

85041 Since the commands run in a separate shell invocation, running in a separate process group with
85042 no controlling terminal, open file descriptors, traps, and priority inherited from the invoking
85043 environment are lost.

85044 Some implementations do not allow substitution of different shells using *SHELL*. System V
85045 systems, for example, have used the login shell value for the user in **/etc/passwd**. To select
85046 reliably another command interpreter, the user must include it as part of the script, such as:

```
85047 $ at 1800
85048 myshell myscript
85049 EOT
85050 job ... at ...
85051 $
```

85052 **EXAMPLES**

85053 1. This sequence can be used at a terminal:

```
85054 at -m 0730 tomorrow
85055 sort < file >outfile
85056 EOT
```

85057 2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a
85058 command procedure (the sequence of output redirection specifications is significant):

```
85059 at now + 1 hour <<!
85060 diff file1 file2 2>&1 >outfile | mailx -s "outfile update" mygroup
85061 !
```

85062 Note that this always sends mail when there has been an attempt to update **outfile** and
85063 the body of the message will be empty unless an error occurred.

85064 3. The following shows how to capture both standard error and standard output:

```
85065 at now + 1 hour <<EOF
```



```
{
    run-batch-processing |
        mailx -s "batch processing output" mygroup
} 2>&1 | mailx -E -s "errors during batch processing" mygroup
EOF
```

4. To have a job reschedule itself, *at* can be invoked from within the at-job. For example, this daily processing script named **my.daily** runs every day (although *crontab* is a more appropriate vehicle for such work):

```
# my.daily runs every day
daily processing
at now tomorrow < my.daily
```

5. The spacing of the three portions of the POSIX locale *timespec* is quite flexible as long as there are no ambiguities. Examples of various times and operand presentation include:

```
at 0815am Jan 24
at 8 :15amjan24
at now "+ 1day"
at 5 pm FRIday
at '17
    utc+
    30minutes'
```

RATIONALE

The *at* utility reads from standard input the commands to be executed at a later time. It may be useful to redirect standard output and standard error within the specified commands.

The **−t** *time* option was added as a new capability to support an internationalized way of specifying a time for execution of the submitted job.

Early proposals added a ``jobname'' concept as a way of giving submitted jobs names that are meaningful to the user submitting them. The historical, system-specified *at_job_id* gives no indication of what the job is. Upon further reflection, it was decided that the benefit of this was not worth the change in historical interface.

The **−q** option historically has been an undocumented option, used mainly by the *batch* utility.

The System V **−m** option was added to provide a method for informing users that an at-job had completed. Otherwise, users are only informed when output to standard error or standard output are not redirected.

The behavior of *at* **<now>** was changed in an early proposal from being unspecified to submitting a job for potentially immediate execution. Historical BSD *at* implementations support this. Historical System V implementations give an error in that case, but a change to the System V versions should have no backwards-compatibility ramifications.

On BSD-based systems, a **−u** *user* option has allowed those with appropriate privileges to access the work of other users. Since this is primarily a system administration feature and is not universally implemented, it has been omitted. Similarly, a specification for the output format for a user with appropriate privileges viewing the queues of other users has been omitted.

The **−f** *file* option from System V is used instead of the BSD method of using the last operand as the pathname. The BSD method is ambiguous—does:

```
at 1200 friday
```

mean the same thing if there is a file named **friday** in the current directory?

The *at_job_id* is composed of a limited character set in historical practice, and it is mandated here to invalidate systems that might try using characters that require shell quoting or that could not be easily parsed by shell scripts.

The *at* utility varies between System V and BSD systems in the way timezones are used. On System V systems, the *TZ* variable affects the at-job submission times and the times displayed for the user. On BSD systems, *TZ* is not taken into account. The BSD behavior is easily achieved with the current specification. If the user wishes to have the timezone default to that of the system, they merely need to issue the *at* command immediately following an unsetting or null assignment to *TZ*. For example:

```
TZ= at noon ...
```

gives the desired BSD result.

While the *yacc*-like grammar specified in the OPERANDS section is lexically unambiguous with respect to the digit strings, a lexical analyzer would probably be written to look for and return digit strings in those cases. The parser could then check whether the digit string returned is a valid *day_number*, *year_number*, and so on, based on the context.

FUTURE DIRECTIONS

None.

SEE ALSO

batch, *crontab*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- If **−m** is not used, the job's standard output and standard error are provided to the user by mail.

The effects of using the **−q** and **−t** options as defined in the IEEE P1003.2b draft standard are specified.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

The *at* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced by the *at* utility.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1307 is applied, changing the *timespec* operand in relation to locales that do not support the 12-hour clock format.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1368 is applied, changing the EXAMPLES section.

Austin Group Defect 1377 is applied, correcting a typographic error in the description of the **−q** option.

Austin Group Defect 1495 is applied, changing the EXIT STATUS and CONSEQUENCES OF ERRORS sections.

NAME

awk — pattern scanning and processing language

SYNOPSIS

```
awk [-F sepstring] [-v assignment]... program [argument...]
awk [-F sepstring] -f progfile [-f progfile]... [-v assignment]...
     [argument...]
```

DESCRIPTION

The *awk* utility shall execute programs written in the *awk* programming language, which is specialized for textual data manipulation. An *awk* program is a sequence of patterns and corresponding actions. When input is read that matches a pattern, the action associated with that pattern is carried out.

Input shall be interpreted as a sequence of records. By default, a record is a line, less its terminating <newline>, but this can be changed by using the **RS** built-in variable. Each record of input shall be matched in turn against each pattern in the program. For each pattern matched, the associated action shall be executed.

The *awk* utility shall interpret each input record as a sequence of fields where, by default, a field is a string of non-<blank> non-<newline> characters. This default <blank> and <newline> field delimiter can be changed by using the **FS** built-in variable or the –**F** *sepstring* option. The *awk* utility shall denote the first field in a record $1, the second $2, and so on. The symbol $0 shall refer to the entire record; setting any other field causes the re-evaluation of $0. Assigning to $0 shall reset the values of all other fields and the **NF** built-in variable.

OPTIONS

The *awk* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–**F** *sepstring* Define the input field separator. This option shall be equivalent to:

```
-v FS=sepstring
```

except that if –**F** *sepstring* and –**v** `FS=`*sepstring* are both used, it is unspecified whether the **FS** assignment resulting from –**F** *sepstring* is processed in command line order or is processed after the last –**v** `FS=`*sepstring*. See the description of the **FS** built-in variable, and how it is used, in the EXTENDED DESCRIPTION section.

–**f** *progfile* Specify the pathname of the file *progfile* containing an *awk* program. A pathname of `'-'` shall denote the standard input. If multiple instances of this option are specified, the concatenation of the files specified as *progfile* in the order specified shall be the *awk* program. The *awk* program can alternatively be specified in the command line as a single argument.

–**v** *assignment*
The application shall ensure that the *assignment* argument is in the same form as an *assignment* operand. The specified variable assignment shall occur prior to executing the *awk* program, including the actions associated with **BEGIN** patterns (if any). Multiple occurrences of this option can be specified.

OPERANDS

The following operands shall be supported:

program If no –**f** option is specified, the first operand to *awk* shall be the text of the *awk* program. The application shall supply the *program* operand as a single argument to *awk*. If the text does not end in a <newline>, *awk* shall interpret the text as if it did.

argument Either of the following two types of *argument* can be intermixed:

file A pathname of a file that contains the input to be read, which is matched against the set of patterns in the program. If no *file* operands or their equivalents, achieved by modifying the *awk* variables **ARGV** and **ARGC**, are specified, or if a *file* operand is `'-'`, the standard input shall be used.

assignment An operand that begins with an <underscore> or alphabetic character from the portable character set (see the table in XBD Section 6.1, on page 117), followed by a sequence of underscores, digits, and alphabetics from the portable character set, followed by the `'='` character, shall specify a variable assignment rather than a pathname. The characters before the `'='` represent the name of an *awk* variable; if that name is an *awk* reserved word (see Grammar, on page 2624) the behavior is undefined. The characters following the <equals-sign> shall be interpreted as if they appeared in the *awk* program preceded and followed by a double-quote (`'"'`) character, as a **STRING** token (see Grammar, on page 2624), except that if the last character is an unescaped <backslash>, it shall be interpreted as a literal <backslash> rather than as the first character of the sequence `"\""`. The variable shall be assigned the value of that **STRING** token and, if appropriate, shall be considered a *numeric string* (see Expressions in awk, on page 2608), the variable shall also be assigned its numeric value. Each such variable assignment shall occur just prior to the processing of the following *file*, if any. Thus, an assignment before the first *file* argument shall be executed after the **BEGIN** actions (if any), while an assignment after the last *file* argument shall occur before the **END** actions (if any). If there are no *file* arguments or their equivalents, achieved by modifying the *awk* variables **ARGV** and **ARGC**, assignments shall be executed before processing the standard input.

STDIN

The standard input shall be used only if no *file* operands or their equivalents, achieved by modifying the *awk* variables **ARGV** and **ARGC**, are specified; or if a *file* operand, or its equivalent, is `'-'`; or if a *progfile* option-argument is `'-'`; see the INPUT FILES section. If the *awk* program contains no actions and no patterns, but is otherwise a valid *awk* program, standard input and any *file* operands shall not be read and *awk* shall exit with a return status of zero.

INPUT FILES

Input files to the *awk* program from any of the following sources shall be text files:

- Any *file* operands or their equivalents, achieved by modifying the *awk* variables **ARGV** and **ARGC**
- Standard input in the absence of any *file* operands, or their equivalents
- Arguments to the **getline** function

Whether the variable **RS** is set to a value other than a <newline> or not, for these files, implementations shall support records terminated with the specified separator up to {LINE_MAX} bytes and may support longer records.

If **−f** *progfile* is specified, the application shall ensure that the files named by each of the *progfile*

option-arguments are text files and their concatenation, in the same order as they appear in the arguments, is an *awk* program.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *awk*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions and in comparisons of string values.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes within regular expressions, the identification of characters as letters, and the mapping of uppercase and lowercase characters for the **toupper** and **tolower** functions.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

LC_NUMERIC
Determine the radix character used when interpreting numeric input, performing conversions between numeric and string values, and formatting numeric output. Regardless of locale, the <period> character (the decimal-point character of the POSIX locale) is the decimal-point character recognized in processing *awk* programs (including assignments in command line arguments).

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

PATH Determine the search path when looking for commands executed by *system*(*expr*), or input and output pipes; see XBD Chapter 8 (on page 167).

In addition, all environment variables shall be visible via the *awk* variable **ENVIRON**.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The nature of the output files depends on the *awk* program.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

The nature of the output files depends on the *awk* program.

EXTENDED DESCRIPTION

Overall Program Structure

An *awk* program is composed of pairs of the form:

```
pattern { action }
```

Either the pattern or the action (including the enclosing brace characters) can be omitted.

A missing pattern shall match any record of input, and a missing action shall be equivalent to:

```
{ print }
```

Execution of the *awk* program shall start by first executing the actions associated with all **BEGIN** patterns in the order they occur in the program. Then each *file* operand (or standard input if no files were specified) shall be processed in turn by reading data from the file until a record separator is seen (<newline> by default). Before the first reference to a field in the record is evaluated, the record shall be split into fields, according to the rules in Regular Expressions (on page 2615), using the value of **FS** that was current at the time the record was read. Each pattern in the program then shall be evaluated in the order of occurrence, and the action associated with each pattern that matches the current record executed. The action for a matching pattern shall be executed before evaluating subsequent patterns. Finally, the actions associated with all **END** patterns shall be executed in the order they occur in the program.

Expressions in awk

Expressions describe computations used in *patterns* and *actions*. In the following table, valid expression operations are given in groups from highest precedence first to lowest precedence last, with equal-precedence operators grouped between horizontal lines. In expression evaluation, where the grammar is formally ambiguous, higher precedence operators shall be evaluated before lower precedence operators. In this table *expr*, *expr1*, *expr2*, and *expr3* represent any expression, while lvalue represents any entity that can be assigned to (that is, on the left side of an assignment operator). The precise syntax of expressions is given in Grammar (on page 2624).

Table 3-1 Expressions in Decreasing Precedence in *awk*

Syntax	Name	Type of Result	Associativity
`( expr )`	Grouping	Type of *expr*	N/A
`$expr`	Field reference	Uninitialized or String	N/A
`lvalue ++`	Post-increment	Numeric	N/A
`lvalue --`	Post-decrement	Numeric	N/A
`++ lvalue`	Pre-increment	Numeric	N/A
`-- lvalue`	Pre-decrement	Numeric	N/A
`expr ^ expr`	Exponentiation	Numeric	Right
`! expr`	Logical not	Numeric	N/A
`+ expr`	Unary plus	Numeric	N/A
`- expr`	Unary minus	Numeric	N/A
`expr * expr`	Multiplication	Numeric	Left
`expr / expr`	Division	Numeric	Left
`expr % expr`	Modulus	Numeric	Left

Syntax	Name	Type of Result	Associativity
expr + *expr*	Addition	Numeric	Left
expr − *expr*	Subtraction	Numeric	Left
expr expr	String concatenation	String	Left
expr < *expr*	Less than	Numeric	None
expr <= *expr*	Less than or equal to	Numeric	None
expr != *expr*	Not equal to	Numeric	None
expr == *expr*	Equal to	Numeric	None
expr > *expr*	Greater than	Numeric	None
expr >= *expr*	Greater than or equal to	Numeric	None
expr ~ *expr*	ERE match	Numeric	None
expr !~ *expr*	ERE non-match	Numeric	None
expr in *array*	Array membership	Numeric	Left
(*index*) in *array*	Multi-dimension array membership	Numeric	Left
expr && *expr*	Logical AND	Numeric	Left
expr || *expr*	Logical OR	Numeric	Left
expr1 ? *expr2* : *expr3*	Conditional expression	Type of selected *expr2* or *expr3*	Right
lvalue ^= *expr*	Exponentiation assignment	Numeric	Right
lvalue %= *expr*	Modulus assignment	Numeric	Right
lvalue *= *expr*	Multiplication assignment	Numeric	Right
lvalue /= *expr*	Division assignment	Numeric	Right
lvalue += *expr*	Addition assignment	Numeric	Right
lvalue −= *expr*	Subtraction assignment	Numeric	Right
lvalue = *expr*	Assignment	Type of *expr*	Right

Each expression shall have either a string value, a numeric value, or both. Except as stated for specific contexts, the value of an expression shall be implicitly converted to the type needed for the context in which it is used. A string value shall be converted to a numeric value either by the equivalent of the following calls to functions defined by the ISO C standard:

```
setlocale(LC_NUMERIC, "");
numeric_value = atof(string_value);
```

or by converting the initial portion of the string to type **double** representation as follows:

> The input string is decomposed into two parts: an initial, possibly empty, sequence of white-space characters (as specified by *isspace*()) and a subject sequence interpreted as a floating-point constant.
>
> The expected form of the subject sequence is an optional '+' or '−' sign, then a non-empty sequence of digits optionally containing a radix character, then an optional exponent part. An exponent part consists of 'e' or 'E', followed by an optional sign, followed by one or more decimal digits.
>
> The sequence starting with the first digit or the radix character (whichever occurs first) is interpreted as a floating constant of the C language, except that the radix character shall be used in place of a <period>, and if neither an exponent part nor a radix character appears, a radix character is assumed to follow the last digit in the string. If the subject sequence begins with a <hyphen-minus>, the value resulting from the conversion is negated.

A numeric value that is exactly equal to the value of an integer (see Section 1.1.2, on page 2457)

shall be converted to a string by the equivalent of a call to the **sprintf** function (see String Functions, on page 2621) with the string `"%d"` as the *fmt* argument and the numeric value being converted as the first and only *expr* argument. Any other numeric value shall be converted to a string by the equivalent of a call to the **sprintf** function with the value of the variable **CONVFMT** as the *fmt* argument and the numeric value being converted as the first and only *expr* argument. The result of the conversion is unspecified if the value of **CONVFMT** is not a floating-point format specification. This volume of POSIX.1-2024 specifies no explicit conversions between numbers and strings. An application can force an expression to be treated as a number by adding zero to it, or can force it to be treated as a string by concatenating the null string (`""`) to it.

A string value shall be considered a *numeric string* if it comes from one of the following:

1. Field variables
2. Input from the *getline*( ) function
3. **FILENAME**
4. **ARGV** array elements
5. **ENVIRON** array elements
6. Array elements created by the *split*( ) function
7. A command line variable assignment
8. Variable assignment from another numeric string variable

and an implementation-dependent condition corresponding to either case (a) or (b) below is met.

a. After the equivalent of the following calls to functions defined by the ISO C standard, *string_value_end* would differ from *string_value*, and any characters before the terminating null character in *string_value_end* would be <blank> characters:

```
char *string_value_end;
setlocale(LC_NUMERIC, "");
numeric_value = strtod (string_value, &string_value_end);
```

b. After all the following conversions have been applied, the resulting string would lexically be recognized as a **NUMBER** token as described by the lexical conventions in Grammar (on page 2624):

   — All leading and trailing <blank> characters are discarded.

   — If the first non-<blank> is `'+'` or `'-'`, it is discarded.

   — Each occurrence of the radix character from the current locale is changed to a <period>.

In case (a) the numeric value of the *numeric string* shall be the value that would be returned by the *strtod*( ) call. In case (b) if the first non-<blank> is `'-'`, the numeric value of the *numeric string* shall be the negation of the numeric value of the recognized **NUMBER** token; otherwise, the numeric value of the *numeric string* shall be the numeric value of the recognized **NUMBER** token. Whether or not a string is a *numeric string* shall be relevant only in contexts where that term is used in this section.

When an expression is used in a Boolean context, if it has a numeric value, a value of zero shall be treated as false and any other value shall be treated as true. Otherwise, a string value of the null string shall be treated as false and any other value shall be treated as true. A Boolean

context shall be one of the following:

- The first subexpression of a conditional expression
- An expression operated on by logical NOT, logical AND, or logical OR
- The second expression of a **for** statement
- The expression of an **if** statement
- The expression of the **while** clause in either a **while** or **do**...**while** statement
- An expression used as a pattern (as in Overall Program Structure)

All arithmetic shall follow the semantics of floating-point arithmetic as specified by the ISO C standard (see Section 1.1.2, on page 2457).

The value of the expression:

expr1 `^` *expr2*

shall be equivalent to the value returned by the ISO C standard function call:

`pow(`*expr1*`,` *expr2*`)`

The expression:

`lvalue ^=` *expr*

shall be equivalent to the ISO C standard expression:

`lvalue = pow(lvalue,` *expr*`)`

except that lvalue shall be evaluated only once. The value of the expression:

expr1 `%` *expr2*

shall be equivalent to the value returned by the ISO C standard function call:

`fmod(`*expr1*`,` *expr2*`)`

The expression:

`lvalue %=` *expr*

shall be equivalent to the ISO C standard expression:

`lvalue = fmod(lvalue,` *expr*`)`

except that lvalue shall be evaluated only once.

Variables and fields shall be set by the assignment statement:

`lvalue =` *expression*

and the type of *expression* shall determine the resulting variable type. The assignment includes the arithmetic assignments (`"+="`, `"-="`, `"*="`, `"/="`, `"%="`, `"^="`, `"++"`, `"--"`) all of which shall produce a numeric result. The left-hand side of an assignment and the target of increment and decrement operators can be one of a variable, an array with index, or a field selector.

The *awk* language supplies arrays that are used for storing numbers or strings. Arrays need not be declared. They shall initially be empty, and their sizes shall change dynamically. The subscripts, or element identifiers, are strings, providing a type of associative array capability. An array name followed by a subscript within square brackets can be used as an lvalue and thus as an expression, as described in the grammar; see Grammar (on page 2624). Unsubscripted array names can be used in only the following contexts:

- A parameter in a function definition or function call
- The **NAME** token following any use of the keyword **in** as specified in the grammar (see Grammar, on page 2624); if the name used in this context is not an array name, the behavior is undefined
- The **NAME** token following the keyword **Delete** without a subscript as specified in the grammar (see Grammar, on page 2624); if the name used in this context is not an array name, the behavior is undefined.

A valid array *index* shall consist of one or more <comma>-separated expressions, similar to the way in which multi-dimensional arrays are indexed in some programming languages. Because *awk* arrays are really one-dimensional, such a <comma>-separated list shall be converted to a single string by concatenating the string values of the separate expressions, each separated from the other by the value of the **SUBSEP** variable. Thus, the following two index operations shall be equivalent:

```
var[expr1, expr2, ... exprn]
var[expr1 SUBSEP expr2 SUBSEP ... SUBSEP exprn]
```

The application shall ensure that a multi-dimensioned *index* used with the **in** operator is parenthesized. The **in** operator, which tests for the existence of a particular array element, shall not cause that element to exist. Any other reference to a nonexistent array element shall automatically create it.

Comparisons (with the `'<'`, `"<="`, `"!="`, `"=="`, `'>'`, and `">="` operators) shall be made numerically:

- if both operands are numeric,
- if one is numeric and the other has a string value that is a numeric string,
- if both have string values that are numeric strings, or
- if one is numeric and the other has the uninitialized value.

Otherwise, operands shall be converted to strings as required and a string comparison shall be made as follows:

- For the `"!="` and `"=="` operators, the strings shall be compared to check if they are identical (not to check if they collate equally).
- For the other operators, the strings shall be compared using the locale-specific collation sequence.

The value of the comparison expression shall be 1 if the relation is true, or 0 if the relation is false.

Variables and Special Variables

Variables can be used in an *awk* program by referencing them. With the exception of function parameters (see User-Defined Functions, on page 2624), they are not explicitly declared. Function parameter names shall be local to the function; all other variable names shall be global. The same name shall not be used as both a function parameter name and as the name of a function or a special *awk* variable. The same name shall not be used both as a variable name with global scope and as the name of a function. The same name shall not be used within the same scope both as a scalar variable and as an array. Uninitialized variables, including scalar variables, array elements, and field variables, shall have an uninitialized value. An uninitialized value shall have both a numeric value of zero and a string value of the empty string. Evaluation

of variables with an uninitialized value, to either string or numeric, shall be determined by the context in which they are used.

Field variables shall be designated by a `'$'` followed by a number or numerical expression. The effect of the field number *expression* evaluating to anything other than a non-negative integer is unspecified; uninitialized variables or string values need not be converted to numeric values in this context. New field variables can be created by assigning a value to them. References to nonexistent fields (that is, fields after $**NF**), shall evaluate to the uninitialized value. Such references shall not create new fields. However, assigning to a nonexistent field (for example, $(**NF**+2)=5) shall increase the value of **NF**; create any intervening fields with the uninitialized value; and cause the value of $0 to be recomputed, with the fields being separated by the value of **OFS**. Each field variable shall have a string value or an uninitialized value when created. Field variables shall have the uninitialized value when created from $0 using **FS** and the variable does not contain any characters. If appropriate, the field variable shall be considered a numeric string (see Expressions in awk, on page 2608).

Implementations shall support the following other special variables that are set by *awk*:

ARGC A number determining when the iteration described for **ARGV** stops. When an *awk* program starts, **ARGC** shall be initialized to the number of elements in the **ARGV** array. **ARGC** can be updated by the *awk* program and by assignment operands. If **ARGC** is set to a value less than 1, the behavior is unspecified. It is unspecified whether alterations to **ARGC** can be made using the **–v** option.

ARGV An array containing, initially, the command name (see Section 2.9.1, on page 2500) used to invoke *awk* in `ARGV[0]` and the command line arguments, if any, excluding options and the *program* operand, in `ARGV[1]` through `ARGV[ARGC−1]`. The elements in **ARGV** can be assigned new values or deleted, and new elements can be added. Note that alterations to **ARGV** cannot be made using either the *assignment* operand or the **–v** option, because an operand with a `'['` before `'='` is treated as a *file* operand, not an *assignment* operand, and applications are required to ensure that the **–v** option-argument has the same form as an *assignment* operand. (See the OPTIONS and OPERANDS sections.)

After processing the **BEGIN** actions, if any, *awk* begins interating over the elements of **ARGV**, processing them as if they were *argument* operands. It shall behave as if the implementation maintains an internal counter that is initialized to 1 and increments by 1 at the end of each iteration. For each iteration, the following shall occur:

- If the internal counter is greater than or equal to the current value of **ARGC** and no *file* operands have been processed, *awk* shall set **FILENAME** to `'−'` and process standard input as if it was given as a file operand. The internal counter shall not be incremented at the end of this iteration.
- Otherwise, if the internal counter is greater than or equal to the current value of **ARGC**, the iterations shall stop and processing of the **END** actions, if any, shall begin. Any **ARGV** elements with index values greater than or equal to **ARGC** shall not be processed as *argument* operands.
- Otherwise, if the element `ARGV[`*internal counter value*`]` does not exist, it is unspecified whether that element is created. No other action shall be taken.
- Otherwise, if `ARGV[`*internal counter value*`]` is a null string, no action shall be taken.

85546 85547 - Otherwise, if `ARGV[`*internal counter value*`]` matches the format of an *assignment* operand (see OPERANDS), *awk* shall process the assignment.

85548 85549 85550 - Otherwise, `ARGV[`*internal counter value*`]` shall be treated as a *file* operand, **FILENAME** shall be set to that value, and the named file, or standard input if the value is `'-'`, shall be processed as an input file.

85551 85552 Since only non-null elements are processed, setting an element of **ARGV** to the null string or deleting it means that it shall not be treated as an *argument* operand.

85553 85554 **CONVFMT** The **printf** format for converting numbers to strings (except for output statements, where **OFMT** is used); `"%.6g"` by default.

85555 85556 85557 85558 85559 85560 85561 **ENVIRON** An array representing the value of the environment, as described in the *exec* functions defined in the System Interfaces volume of POSIX.1-2024. The indices of the array shall be strings consisting of the names of the environment variables, and the value of each array element shall be a string consisting of the value of that variable. If appropriate, the environment variable shall be considered a *numeric string* (see Expressions in awk, on page 2608); the array element shall also have its numeric value.

85562 85563 85564 85565 85566 85567 In all cases where the behavior of *awk* is affected by environment variables (including the environment of any commands that *awk* executes via the **system** function or via pipeline redirections with the **print** statement, the **printf** statement, or the **getline** function), the environment used shall be the environment at the time *awk* began executing; it is implementation-defined whether any modification of **ENVIRON** affects this environment.

85568 85569 85570 85571 **FILENAME** The pathname used to open the current input file, or `'-'` if the file is standard input. Inside a **BEGIN** action **FILENAME** shall be unset. Inside an **END** action the value shall be the name of the last input file processed. If an application changes the value of **FILENAME**, the results are unspecified.

85572 85573 85574 **FNR** The ordinal number of the current record in the current file. Inside a **BEGIN** action the value shall be zero. Inside an **END** action the value shall be the number of the last record processed in the last file processed.

85575 **FS** Input field separator regular expression; a <space> by default.

85576 85577 85578 85579 85580 **NF** The number of fields in the current record. Inside a **BEGIN** action, the use of **NF** is undefined unless a **getline** function without a *var* argument is executed previously. Inside an **END** action, **NF** shall retain the value it had for the last record read, unless a subsequent, redirected, **getline** function without a *var* argument is performed prior to entering the **END** action.

85581 85582 85583 85584 **NR** The ordinal number of the current record from the start of input. Inside a **BEGIN** action the value shall be zero. Inside an **END** action the value shall be the number of the last record processed. Records skipped by the **nextfile** statement shall not be included.

85585 85586 85587 **OFMT** The **printf** format for converting numbers to strings in output statements (see Output Statements, on page 2619); `"%.6g"` by default. The result of the conversion is unspecified if the value of **OFMT** is not a floating-point format specification.

85588 **OFS** The **print** statement output field separator; <space> by default.

85589 **ORS** The **print** statement output record separator; a <newline> by default.

85590 **RLENGTH** The length of the string matched by the **match** function.

85591 **RS** The first character of the string value of **RS** shall be the input record separator; a
85592 <newline> by default. If **RS** contains more than one character, the results are
85593 unspecified. If **RS** is null, then records are separated by sequences consisting of a
85594 <newline> plus one or more blank lines, leading or trailing blank lines shall not
85595 result in empty records at the beginning or end of the input, and a <newline> shall
85596 always be a field separator, no matter what the value of **FS** is.

85597 **RSTART** The starting position of the string matched by the **match** function, numbering from
85598 1. This shall always be equivalent to the return value of the **match** function.

85599 **SUBSEP** The subscript separator string for multi-dimensional arrays; the default value is
85600 implementation-defined.

85601 **Regular Expressions**

85602 The *awk* utility shall make use of the extended regular expression notation (see XBD Section 9.4,
85603 on page 187) except that it shall allow the use of C-language conventions for escaping special
85604 characters within the EREs, as specified in the table in XBD Chapter 5 (on page 113) for `'\\'`,
85605 `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'` and in the following table for other sequences;
85606 these escape sequences shall be recognized both inside and outside bracket expressions. Note
85607 that records need not be separated by <newline> characters and string constants can contain
85608 <newline> characters, so even the `"\n"` sequence is valid in *awk* EREs. Using a <slash>
85609 character within the lexical token **ERE** (except as one of the two delimiters) requires the escaping
85610 shown in the following table.

85611 **Table 3-2** Escape Sequences in *awk*

Escape Sequence	Description	Meaning
`\"`	<backslash> <quotation-mark>	In the lexical token **STRING**, <quotation-mark> character. Otherwise undefined.
`\/`	<backslash> <slash>	In the lexical token **ERE**, <slash> character. Otherwise undefined.
`\ddd`	A <backslash> character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined. If the digits produce a value greater than octal 377, the behavior is undefined.	The character whose encoding is represented by the one, two, or three-digit octal integer. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading <backslash> for each byte.
`\.`, `\[`, `\(`, `\*`, `\+`, `\?`, `\{`, `|`, `\^`, `\$`	A <backslash> character followed by a character that has a special meaning in EREs (see XBD Section 9.4), other than <backslash>.	In the lexical token **ERE** when not inside a bracket expression, the sequence shall represent itself. Otherwise undefined.
`\\`	Two <backslash> characters.	In the lexical token **ERE**, the sequence shall represent itself. In the lexical token **STRING**, it shall represent a single <backslash>.
`\c`	A <backslash> character followed by any character not described in this table or in the table in XBD Chapter 5 (on page 113) (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`).	Undefined

A regular expression can be matched against a specific field or string by using one of the two regular expression matching operators, `'~'` and `"!~"`. These operators shall interpret their right-hand operand as a regular expression and their left-hand operand as a string. If the regular expression matches the string, the `'~'` expression shall evaluate to a value of 1, and the `"!~"` expression shall evaluate to a value of 0. (The regular expression matching operation is as defined by the term matched in XBD Section 9.1 (on page 179), where a match occurs on any part of the string unless the regular expression is limited with the <circumflex> or <dollar-sign> special characters.) If the regular expression does not match the string, the `'~'` expression shall evaluate to a value of 0, and the `"!~"` expression shall evaluate to a value of 1. If the right-hand operand is any expression other than the lexical token **ERE**, the string value of the expression shall be interpreted as an extended regular expression, including the escape conventions described above. Note that these escape conventions shall also be applied in determining the value of a string literal (the lexical token **STRING**), and thus shall be applied a second time when a string literal is used in this context.

When an **ERE** token appears as an expression in any context other than as the right-hand of the `'~'` or `"!~"` operator or as one of the built-in function arguments described below, the value of the resulting expression shall be the equivalent of:

```
$0 ~ /ere/
```


The *ere* argument to the **gsub**, **match**, **sub** functions, and the *fs* argument to the **split** function (see String Functions, on page 2621) shall be interpreted as extended regular expressions. These can be either **ERE** tokens or arbitrary expressions, and shall be interpreted in the same manner as the right-hand side of the `'~'` or `"!~"` operator.

An extended regular expression can be used to separate fields by assigning a string containing the expression to the built-in variable **FS**, either directly or as a consequence of using the **–F** *sepstring* option. The default value of the **FS** variable shall be a single <space>. The following describes **FS** behavior:

1. If **FS** is a null string, the behavior is unspecified.
2. If **FS** is a single character:
   a. If **FS** is <space>, skip leading and trailing <blank> and <newline> characters; fields shall be delimited by sets of one or more <blank> or <newline> characters.
   b. Otherwise, if **FS** is any other character *c*, fields shall be delimited by each single occurrence of *c*.
3. Otherwise, the string value of **FS** shall be considered to be an extended regular expression. Each occurrence of a sequence of one or more characters matching the extended regular expression shall delimit fields.

When ERE matching is performed against input records; that is, the match is against $0 and the current value of $0 resulted from processing an input record, record separator characters (the first character of the value of the variable **RS**, <newline> by default) cannot be embedded in the expression, and no expression shall match the record separator character. If the record separator is not <newline>, <newline> characters embedded in the expression can be matched. When ERE matching is not performed against input records, it shall be based on text strings; any character (including <newline> and the record separator) can be embedded in the pattern, and an appropriate pattern shall match any character. However, in all *awk* ERE matching, the use of one or more NUL characters in the pattern, input record, or text string produces undefined results.

Patterns

A *pattern* is any valid *expression*, a range specified by two expressions separated by a comma, or one of the two special patterns **BEGIN** or **END**.

Special Patterns

The *awk* utility shall recognize two special patterns, **BEGIN** and **END**. Each **BEGIN** pattern shall be matched once and its associated action executed before the first record of input is read—except possibly by use of the **getline** function (see Input/Output and General Functions, on page 2623) in a prior **BEGIN** action—and before command line assignment is done. Each **END** pattern shall be matched once and its associated action executed after the last record of input has been read, or if there is no further input file to process following a **nextfile** statement. These two patterns shall have associated actions.

BEGIN and **END** shall not combine with other patterns. Multiple **BEGIN** and **END** patterns shall be allowed. The actions associated with the **BEGIN** patterns shall be executed in the order specified in the program, as are the **END** actions. An **END** pattern can precede a **BEGIN** pattern in a program.

If an *awk* program consists of only actions with the pattern **BEGIN**, and the **BEGIN** action contains no **getline** function, *awk* shall exit without reading its input when the last statement in the last **BEGIN** action is executed. If an *awk* program consists of only actions with the pattern

85702 **END** or only actions with the patterns **BEGIN** and **END**, the input shall be read before the
85703 statements in the **END** actions are executed.

85704 **Expression Patterns**

85705 An expression pattern shall be evaluated as if it were an expression in a Boolean context. If the
85706 result is true, the pattern shall be considered to match, and the associated action (if any) shall be
85707 executed. If the result is false, the action shall not be executed.

85708 **Pattern Ranges**

85709 A pattern range consists of two expressions separated by a comma; in this case, the action shall
85710 be performed for all records between a match of the first expression and the following match of
85711 the second expression, inclusive. At this point, the pattern range can be repeated starting at
85712 input records subsequent to the end of the matched range.

85713 **Actions**

85714 An action is a sequence of statements as shown in the grammar in Grammar (on page 2624).
85715 Any single statement can be replaced by a statement list enclosed in curly braces. The
85716 application shall ensure that statements in a statement list are separated by <newline> or
85717 <semicolon> characters. Statements in a statement list shall be executed sequentially in the order
85718 that they appear.

85719 The *expression* acting as the conditional in an **if** statement shall be evaluated and if it is non-zero
85720 or non-null, the following statement shall be executed; otherwise, if **else** is present, the statement
85721 following the **else** shall be executed.

85722 The **if**, **while**, **do**...**while**, **for**, **break**, and **continue** statements are based on the ISO C standard
85723 (see Section 1.1.2, on page 2457), except that the Boolean expressions shall be treated as
85724 described in Expressions in awk (on page 2608), and except in the case of:

85725 `for (`*`variable`*` in `*`array`*`)`

85726 which shall iterate, assigning each *index* of *array* to *variable* in an unspecified order. The results of
85727 adding new elements to *array* within such a **for** loop are undefined. If a **break** or **continue**
85728 statement occurs outside of a loop, the behavior is undefined.

85729 The **delete** statement shall remove either a specified individual array element or, if no element is
85730 specified, all array elements. Thus, the following code:

85731 ```
85732 for (index in array)
        delete array[index]
```

85733 is equivalent to:

85734 ```
delete array
```

85735 Both delete all elements of the array.

85736 The **next** statement shall cause all further processing of the current input record to be
85737 abandoned. The behavior is undefined if a **next** statement appears or is invoked in a **BEGIN** or
85738 **END** action.

85739 The **nextfile** statement shall cause all further processing of the current input file to be
85740 abandoned. The behavior is undefined if a **nextfile** statement appears or is invoked in a **BEGIN**
85741 or **END** action, or in a user-defined function.

85742 The **exit** statement shall invoke all **END** actions in the order in which they occur in the program

source and then terminate the program without reading further input. An **exit** statement inside an **END** action shall terminate the program without further execution of **END** actions. If an expression is specified in an **exit** statement, its numeric value shall be the exit status of *awk*, unless subsequent errors are encountered or a subsequent **exit** statement with an expression is executed.

Output Statements

Both **print** and **printf** statements shall write to standard output by default. The output shall be written to the location specified by *output_redirection* if one is supplied, as follows:

```
> expression
>> expression
| expression
```

In all cases, the *expression* shall be evaluated to produce a string that is used as a pathname into which to write (for `'>'` or `">>"`) or as a command to be executed (for `'|'`). Using the first two forms, if the file of that name is not currently open, it shall be opened, creating it if necessary and using the first form, truncating the file. The output then shall be appended to the file. As long as the file remains open, subsequent calls in which *expression* evaluates to the same string value shall simply append output to the file. The file remains open until the **close** function (see Input/Output and General Functions, on page 2623) is called with an expression that evaluates to the same string value.

The third form shall write output onto a stream piped to the input of a command. The stream shall be created if no stream is currently open with the value of *expression* as its command name. The stream created shall be equivalent to one created by a call to the *popen*( ) function defined in the System Interfaces volume of POSIX.1-2024 with the value of *expression* as the *command* argument and a value of *w* as the *mode* argument. As long as the stream remains open, subsequent calls in which *expression* evaluates to the same string value shall write output to the existing stream. The stream shall remain open until the **close** function (see Input/Output and General Functions, on page 2623) is called with an expression that evaluates to the same string value. At that time, the stream shall be closed as if by a call to the *pclose*( ) function defined in the System Interfaces volume of POSIX.1-2024.

As described in detail by the grammar in Grammar (on page 2624), these output statements shall take a <comma>-separated list of *expression*s referred to in the grammar by the non-terminal symbols **expr_list**, **print_expr_list**, or **print_expr_list_opt**. This list is referred to here as the *expression list*, and each member is referred to as an *expression argument*.

The **print** statement shall write the value of each expression argument onto the indicated output stream separated by the current output field separator (see variable **OFS** above), and terminated by the output record separator (see variable **ORS** above). All expression arguments shall be taken as strings, being converted if necessary; this conversion shall be as described in Expressions in awk (on page 2608), with the exception that the **printf** format in **OFMT** shall be used instead of the value in **CONVFMT**. An empty expression list shall stand for the whole input record ($0).

The **printf** statement shall produce output based on a notation similar to the File Format Notation used to describe file formats in this volume of POSIX.1-2024 (see XBD Chapter 5, on page 113). Output shall be produced as specified with the first *expression* argument as the string *format* and subsequent *expression* arguments as the strings *arg1* to *argn*, inclusive, with the following exceptions:

85788 1. The *format* shall be an actual character string rather than a graphical representation.
85789 Therefore, it cannot contain empty character positions. The <space> in the *format* string,
85790 in any context other than a *flag* of a conversion specification, shall be treated as an
85791 ordinary character that is copied to the output.

85792 2. If the character set contains a `'Δ'` character and that character appears in the *format*
85793 string, it shall be treated as an ordinary character that is copied to the output.

85794 3. The *escape sequences* beginning with a <backslash> character shall be treated as sequences
85795 of ordinary characters that are copied to the output. Note that these same sequences shall
85796 be interpreted lexically by *awk* when they appear in literal strings, but they shall not be
85797 treated specially by the **printf** statement.

85798 4. A *field width* or *precision* can be specified as the `'*'` character instead of a digit string. In
85799 this case the next argument from the expression list shall be fetched and its numeric value
85800 taken as the field width or precision.

85801 5. The implementation shall not precede or follow output from the `d` or `u` conversion
85802 specifier characters with <blank> characters not specified by the *format* string.

85803 6. The implementation shall not precede output from the `o` conversion specifier character
85804 with leading zeros not specified by the *format* string.

85805 7. For the `c` conversion specifier character: if the argument has a numeric value, the
85806 character whose encoding is that value shall be output. If the value is zero or is not the
85807 encoding of any character in the character set, the behavior is undefined. If the argument
85808 does not have a numeric value, the first character of the string value shall be output; if the
85809 string does not contain any characters, the behavior is undefined.

85810 8. For each conversion specification that consumes an argument, the next expression
85811 argument shall be evaluated. With the exception of the `c` conversion specifier character,
85812 the value shall be converted (according to the rules specified in Expressions in awk, on
85813 page 2608) to the appropriate type for the conversion specification.

85814 9. If there are insufficient expression arguments to satisfy all the conversion specifications in
85815 the *format* string, the behavior is undefined.

85816 10. If any character sequence in the *format* string begins with a `'%'` character, but does not
85817 form a valid conversion specification, the behavior is unspecified.

85818 Both **print** and **printf** can output at least {LINE_MAX} bytes.

85819 **Functions**

85820 The *awk* language has a variety of built-in functions: arithmetic, string, input/output, and
85821 general.

85822 Function parameters, if present, can be either scalars or arrays; the behavior is undefined if an
85823 array name is passed as a parameter that the function uses as a scalar, or if a scalar expression is
85824 passed as a parameter that the function uses as an array. Function parameters shall be passed by
85825 value if scalar and by reference if array name.

85826 **Arithmetic Functions**

85827 The arithmetic functions, except for **int**, shall be based on the ISO C standard (see Section 1.1.2,
85828 on page 2457). The behavior is undefined in cases where the ISO C standard specifies that an
85829 error be returned or that the behavior is undefined. Although the grammar (see Grammar, on
85830 page 2624) permits built-in functions to appear with no arguments or parentheses, unless the
85831 argument or parentheses are indicated as optional in the following list (by displaying them
85832 within the "`[]`" brackets), such use is undefined.

85833 **atan2**(*y*,*x*) Return arctangent of y/x in radians in the range $[-\pi,\pi]$.

85834 **cos**(*x*) Return cosine of *x*, where *x* is in radians.

85835 **sin**(*x*) Return sine of *x*, where *x* is in radians.

85836 **exp**(*x*) Return the exponential function of *x*.

85837 **log**(*x*) Return the natural logarithm of *x*.

85838 **sqrt**(*x*) Return the square root of *x*.

85839 **int**(*x*) Return the argument truncated to an integer. Truncation shall be toward 0 when
85840 $x>0$.

85841 **rand**( ) Return a floating point pseudo-random number *n*, such that $0 \le n < 1$.

85842 **srand**([*expr*]) Set the seed value for **rand** to *expr* or use the seconds since the Epoch if *expr* is
85843 omitted. The previous seed value shall be returned. The behavior is unspecified if
85844 *expr* is not an integer expression or if the value of *expr* is not within the range 0
85845 through $2^{31}-1$ (2 147 483 647), inclusive. The initial seed value is unspecified if **rand**
85846 is called without calling **srand** first. The **srand** function uses the argument as a
85847 seed for a new sequence of pseudo-random numbers to be returned by subsequent
85848 calls to **rand**. If **srand** is then called with the same seed value, the sequence of
85849 pseudo-random numbers shall be repeated.

85850 **String Functions**

85851 The string functions in the following list shall be supported. Although the grammar (see
85852 Grammar, on page 2624) permits built-in functions to appear with no arguments or parentheses,
85853 unless the argument or parentheses are indicated as optional in the following list (by displaying
85854 them within the "`[]`" brackets), such use is undefined.

85855 **gsub**(*ere*, *repl*[, *in*])
85856 Behave like **sub** (see below), except that it shall replace all occurrences of the
85857 regular expression (like the *ed* utility global substitute) in $0 or in the *in* argument,
85858 when specified.

85859 **index**(*s*, *t*) Return the position, in characters, numbering from 1, in string *s* where string *t* first
85860 occurs, or zero if it does not occur at all.

85861 **length**[([*arg*])]
85862 If *arg* is an array, return the number of elements in the array; otherwise, return the
85863 length, in characters, of *arg* taken as a string, or of the whole record, $0, if there is
85864 no argument.

85865 **match**(*s*, *ere*) Return the position, in characters, numbering from 1, in string *s* where the
85866 extended regular expression *ere* occurs, or zero if it does not occur at all. RSTART
85867 shall be set to the starting position (which is the same as the returned value), zero
85868 if no match is found; RLENGTH shall be set to the length of the matched string, –1

85869 if no match is found.

85870 **split**(*s*, *a*[, *fs*])
85871 Split the string *s* into array elements *a*[1], *a*[2], ..., *a*[*n*], and return *n*. All elements
85872 of the array shall be deleted before the split is performed. The separation shall be
85873 done with the ERE *fs* or with the field separator **FS** if *fs* is not given. Each array
85874 element shall have a string value when created and, if appropriate, the array
85875 element shall be considered a numeric string (see Expressions in awk, on page
85876 2608). The effect of a null string as the value of *fs* is unspecified.

85877 **sprintf**(*fmt*, *expr*, *expr*, ...)
85878 Format the expressions according to the **printf** format given by *fmt* and return the
85879 resulting string.

85880 **sub(***ere*, *repl*[, *in*])
85881 Substitute the string *repl* in place of the first instance of the extended regular
85882 expression *ERE* in string *in* and return the number of substitutions. An
85883 <ampersand> (`'&'`) appearing in the string *repl* shall be replaced by the string
85884 from *in* that matches the ERE. An <ampersand> preceded with a <backslash> shall
85885 be interpreted as the literal <ampersand> character. An occurrence of two
85886 consecutive <backslash> characters shall be interpreted as just a single literal
85887 <backslash> character. Any other occurrence of a <backslash> (for example,
85888 preceding any other character) shall be treated as a literal <backslash> character.
85889 Note that if *repl* is a string literal (the lexical token **STRING**; see Grammar, on page
85890 2624), the handling of the <ampersand> character occurs after any lexical
85891 processing, including any lexical <backslash>-escape sequence processing. If *in* is
85892 specified and it is not an lvalue (see Expressions in awk, on page 2608), the
85893 behavior is undefined. If *in* is omitted, *awk* shall use the current record ($0) in its
85894 place.

85895 **substr**(*s*, *m*[, *n*])
85896 Return the at most *n*-character substring of *s* that begins at position *m*, numbering
85897 from 1. If *n* is omitted, or if *n* specifies more characters than are left in the string,
85898 the length of the substring shall be limited by the length of the string *s*.

85899 **tolower**(*s*) Return a string based on the string *s*. Each character in *s* that is an uppercase letter
85900 specified to have a **tolower** mapping by the *LC_CTYPE* category of the current
85901 locale shall be replaced in the returned string by the lowercase letter specified by
85902 the mapping. Other characters in *s* shall be unchanged in the returned string.

85903 **toupper**(*s*) Return a string based on the string *s*. Each character in *s* that is a lowercase letter
85904 specified to have a **toupper** mapping by the *LC_CTYPE* category of the current
85905 locale is replaced in the returned string by the uppercase letter specified by the
85906 mapping. Other characters in *s* are unchanged in the returned string.

85907 All of the preceding functions that take *ERE* as a parameter expect a pattern or a string valued
85908 expression that is a regular expression as defined in Regular Expressions (on page 2615).

85909 **Input/Output and General Functions**

85910 The input/output and general functions are:

85911 **close**(*expression*)

85912 Close the file or pipe opened by a **print** or **printf** statement or a call to **getline** with
85913 the same string-valued *expression*. The limit on the number of open *expression*
85914 arguments is implementation-defined. If the close was successful, the function
85915 shall return zero; otherwise, it shall return non-zero.

85916 **fflush**([*expression*])

85917 Write any unwritten data to the file or piped stream opened by a **print** or **printf**
85918 statement with the same string-valued *expression*. If no argument, or if *expression*
85919 evaluates to the null string, then write all such data for all such open files and
85920 piped streams, and standard output.

85921 If **fflush** is successful, it shall return 0; otherwise, it shall return non-zero.

85922 *expression* | **getline** [*var*]

85923 Read a record of input from a stream piped from the output of a command. The
85924 stream shall be created if no stream is currently open with the value of *expression* as
85925 its command name. The stream created shall be equivalent to one created by a call
85926 to the *popen*( ) function with the value of *expression* as the *command* argument and a
85927 value of *r* as the *mode* argument. As long as the stream remains open, subsequent
85928 calls in which *expression* evaluates to the same string value shall read subsequent
85929 records from the stream. The stream shall remain open until the **close** function is
85930 called with an expression that evaluates to the same string value. At that time, the
85931 stream shall be closed as if by a call to the *pclose*( ) function. If *var* is omitted, $0 and
85932 **NF** shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered
85933 a numeric string (see Expressions in awk, on page 2608).

85934 The **getline** operator can form ambiguous constructs when there are
85935 unparenthesized operators (including concatenate) to the left of the `'|'` (to the
85936 beginning of the expression containing **getline**). In the context of the `'$'` operator,
85937 `'|'` shall behave as if it had a lower precedence than `'$'`. The result of evaluating
85938 other operators is unspecified, and conforming applications shall parenthesize
85939 properly all such usages.

85940 **getline** Set $0 to the next input record from the current input file. This form of **getline** shall
85941 set the **NF**, **NR**, and **FNR** variables.

85942 **getline** *var* Set variable *var* to the next input record from the current input file and, if
85943 appropriate, *var* shall be considered a numeric string (see Expressions in awk, on
85944 page 2608). This form of **getline** shall set the **FNR** and **NR** variables.

85945 **getline** [*var*] **<** *expression*

85946 Read the next record of input from a named file. The *expression* shall be evaluated
85947 to produce a string that is used as a pathname. If the file of that name is not
85948 currently open, it shall be opened. As long as the stream remains open, subsequent
85949 calls in which *expression* evaluates to the same string value shall read subsequent
85950 records from the file. The file shall remain open until the **close** function is called
85951 with an expression that evaluates to the same string value. If *var* is omitted, $0 and
85952 **NF** shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered
85953 a numeric string (see Expressions in awk, on page 2608).

85954 The **getline** operator can form ambiguous constructs when there are
85955 unparenthesized binary operators (including concatenate) to the right of the `'<'`

85956 (up to the end of the expression containing the **getline**). The result of evaluating
85957 such a construct is unspecified, and conforming applications shall parenthesize
85958 properly all such usages.

85959 **system**(*expression*)
85960 Execute the command given by *expression* in a manner equivalent to the *system*()
85961 function defined in the System Interfaces volume of POSIX.1-2024 and return the
85962 exit status of the command.

85963 All forms of **getline** shall return 1 for successful input, zero for end-of-file, and –1 for an error.

85964 Where strings are used as the name of a file or pipeline, the application shall ensure that the
85965 strings are textually identical. The terminology ``same string value'' implies that ``equivalent
85966 strings'', even those that differ only by <space> characters, represent different files.

85967 **User-Defined Functions**

85968 The *awk* language also provides user-defined functions. Such functions can be defined as:

85969 `function` *`name`*`([`*`parameter`*`, ...]) { `*`statements`*` }`

85970 A function can be referred to anywhere in an *awk* program; in particular, its use can precede its
85971 definition. The scope of a function is global.

85972 The number of parameters in the function definition need not match the number of parameters
85973 in the function call. Excess formal parameters can be used as local variables. If fewer arguments
85974 are supplied in a function call than are in the function definition, the extra parameters that are
85975 used in the function body as scalars shall evaluate to the uninitialized value until they are
85976 otherwise initialized, and the extra parameters that are used in the function body as arrays shall
85977 be treated as uninitialized arrays where each element evaluates to the uninitialized value until
85978 otherwise initialized.

85979 When invoking a function, no white space can be placed between the function name and the
85980 opening parenthesis. Function calls can be nested and recursive calls can be made upon
85981 functions. Upon return from any nested or recursive function call, the values of all of the calling
85982 function's parameters shall be unchanged, except for array parameters passed by reference. The
85983 **return** statement can be used to return a value. If a **return** statement appears outside of a
85984 function definition, the behavior is undefined.

85985 In the function definition, <newline> characters shall be optional before the opening brace and
85986 after the closing brace. Function definitions can appear anywhere in the program where a
85987 *pattern-action* pair is allowed.

85988 **Grammar**

85989 The grammar in this section and the lexical conventions in the following section shall together
85990 describe the syntax for *awk* programs. The general conventions for this style of grammar are
85991 described in Section 1.3 (on page 2461). A valid program can be represented as the non-terminal
85992 symbol *program* in the grammar. This formal syntax shall take precedence over the preceding
85993 text syntax description.

```
85994 %token NAME NUMBER STRING ERE
85995 %token FUNC_NAME   /* Name followed by '(' without white space. */

85996 /* Keywords */
85997 %token       Begin   End
85998 /*          'BEGIN' 'END'                                  */
```



```
85999       %token       Break   Continue   Delete   Do   Else
86000       /*          'break' 'continue' 'delete' 'do' 'else'      */

86001       %token       Exit   For   Function   If   In   Next
86002       /*          'exit' 'for' 'function' 'if' 'in' 'next'     */

86003       %token       Nextfile   Print   Printf   Return   While
86004       /*          'nextfile' 'print' 'printf' 'return' 'while' */

86005       /* Reserved function names */
86006       %token BUILTIN_FUNC_NAME
86007                   /* One token for the following:
86008                    * atan2 cos sin exp log sqrt int rand srand
86009                    * gsub index length match split sprintf sub
86010                    * substr tolower toupper close fflush system
86011                    */
86012       %token GETLINE
86013                   /* Syntactically different from other built-ins. */

86014       /* Two-character tokens. */
86015       %token ADD_ASSIGN SUB_ASSIGN MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN POW_ASSIGN
86016       /*     '+='       '-='       '*='       '/='       '%='       '^=' */

86017       %token OR   AND  NO_MATCH   EQ   LE   GE   NE   INCR  DECR  APPEND
86018       /*     '||' '&&' '!~' '==' '<=' '>=' '!=' '++'  '--'  '>>'   */

86019       /* One-character tokens. */
86020       %token '{' '}' '(' ')' '[' ']' ',' ';' NEWLINE
86021       %token '+' '-' '*' '%' '^' '!' '>' '<' '|' '?' ':' '~' '$' '='

86022       %start program
86023       %%

86024       program          : item_list
86025                        | item_list item
86026                        ;

86027       item_list        : /* empty */
86028                        | item_list item terminator
86029                        ;

86030       item             : action
86031                        | pattern action
86032                        | normal_pattern
86033                        | Function NAME      '(' param_list_opt ')'
86034                             newline_opt action
86035                        | Function FUNC_NAME '(' param_list_opt ')'
86036                             newline_opt action
86037                        ;

86038       param_list_opt   : /* empty */
86039                        | param_list
86040                        ;

86041       param_list       : NAME
86042                        | param_list ',' NAME
86043                        ;
```



```
86044       pattern          : normal_pattern
86045                        | special_pattern
86046                        ;
86047       normal_pattern   : expr
86048                        | expr ',' newline_opt expr
86049                        ;
86050       special_pattern  : Begin
86051                        | End
86052                        ;
86053       action           : '{' newline_opt                              '}'
86054                        | '{' newline_opt terminated_statement_list   '}'
86055                        | '{' newline_opt unterminated_statement_list '}'
86056                        ;
86057       terminator       : terminator NEWLINE
86058                        |            ';'
86059                        |            NEWLINE
86060                        ;
86061       terminated_statement_list : terminated_statement
86062                        | terminated_statement_list terminated_statement
86063                        ;
86064       unterminated_statement_list : unterminated_statement
86065                        | terminated_statement_list unterminated_statement
86066                        ;
86067       terminated_statement : action newline_opt
86068                        | If '(' expr ')' newline_opt terminated_statement
86069                        | If '(' expr ')' newline_opt terminated_statement
86070                             Else newline_opt terminated_statement
86071                        | While '(' expr ')' newline_opt terminated_statement
86072                        | For '(' simple_statement_opt ';'
86073                           expr_opt ';' simple_statement_opt ')' newline_opt
86074                           terminated_statement
86075                        | For '(' NAME In NAME ')' newline_opt
86076                           terminated_statement
86077                        | ';' newline_opt
86078                        | terminatable_statement NEWLINE newline_opt
86079                        | terminatable_statement ';'     newline_opt
86080                        ;
86081       unterminated_statement : terminatable_statement
86082                        | If '(' expr ')' newline_opt unterminated_statement
86083                        | If '(' expr ')' newline_opt terminated_statement
86084                           Else newline_opt unterminated_statement
86085                        | While '(' expr ')' newline_opt unterminated_statement
86086                        | For '(' simple_statement_opt ';'
86087                         expr_opt ';' simple_statement_opt ')' newline_opt
86088                           unterminated_statement
86089                        | For '(' NAME In NAME ')' newline_opt
86090                           unterminated_statement
86091                        ;
```



```
86092       terminatable_statement : simple_statement
86093                              | Break
86094                              | Continue
86095                              | Next
86096                              | Nextfile
86097                              | Exit expr_opt
86098                              | Return expr_opt
86099                              | Do newline_opt terminated_statement While '(' expr ')'
86100                              ;
86101       simple_statement_opt : /* empty */
86102                              | simple_statement
86103                              ;
86104       simple_statement : Delete NAME '[' expr_list ']'
86105                        | Delete NAME
86106                        | expr
86107                        | print_statement
86108                        ;
86109       print_statement  : simple_print_statement
86110                        | simple_print_statement output_redirection
86111                        ;
86112       simple_print_statement : Print  print_expr_list_opt
86113                              | Print  '(' multiple_expr_list ')'
86114                              | Printf print_expr_list
86115                              | Printf '(' multiple_expr_list ')'
86116                              ;
86117       output_redirection : '>'    expr
86118                          | APPEND expr
86119                          | '|'    expr
86120                          ;
86121       expr_list_opt    : /* empty */
86122                          | expr_list
86123                          ;
86124       expr_list        : expr
86125                          | multiple_expr_list
86126                          ;
86127       multiple_expr_list : expr ',' newline_opt expr
86128                          | multiple_expr_list ',' newline_opt expr
86129                          ;
86130       expr_opt         : /* empty */
86131                          | expr
86132                          ;
86133       expr             : unary_expr
86134                          | non_unary_expr
86135                          ;
86136       unary_expr       : '+' expr
86137                          | '-' expr
```



```
                        | unary_expr '^'      expr
                        | unary_expr '*'      expr
                        | unary_expr '/'      expr
                        | unary_expr '%'      expr
                        | unary_expr '+'      expr
                        | unary_expr '-'      expr
                        | unary_expr          non_unary_expr
                        | unary_expr '<'      expr
                        | unary_expr LE       expr
                        | unary_expr NE       expr
                        | unary_expr EQ       expr
                        | unary_expr '>'      expr
                        | unary_expr GE       expr
                        | unary_expr '~'      expr
                        | unary_expr NO_MATCH expr
                        | unary_expr In NAME
                        | unary_expr AND newline_opt expr
                        | unary_expr OR  newline_opt expr
                        | unary_expr '?' expr ':' expr
                        | unary_input_function
                        ;
    non_unary_expr      : '(' expr ')'
                        | '!' expr
                        | non_unary_expr '^'      expr
                        | non_unary_expr '*'      expr
                        | non_unary_expr '/'      expr
                        | non_unary_expr '%'      expr
                        | non_unary_expr '+'      expr
                        | non_unary_expr '-'      expr
                        | non_unary_expr          non_unary_expr
                        | non_unary_expr '<'      expr
                        | non_unary_expr LE       expr
                        | non_unary_expr NE       expr
                        | non_unary_expr EQ       expr
                        | non_unary_expr '>'      expr
                        | non_unary_expr GE       expr
                        | non_unary_expr '~'      expr
                        | non_unary_expr NO_MATCH expr
                        | non_unary_expr In NAME
                        | '(' multiple_expr_list ')' In NAME
                        | non_unary_expr AND newline_opt expr
                        | non_unary_expr OR  newline_opt expr
                        | non_unary_expr '?' expr ':' expr
                        | NUMBER
                        | STRING
                        | lvalue
                        | ERE
                        | lvalue INCR
                        | lvalue DECR
                        | INCR lvalue
                        | DECR lvalue
```



```
                        | lvalue POW_ASSIGN expr
                        | lvalue MOD_ASSIGN expr
                        | lvalue MUL_ASSIGN expr
                        | lvalue DIV_ASSIGN expr
                        | lvalue ADD_ASSIGN expr
                        | lvalue SUB_ASSIGN expr
                        | lvalue '=' expr
                        | FUNC_NAME '(' expr_list_opt ')'
                           /* no white space allowed before '(' */
                        | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
                        | BUILTIN_FUNC_NAME
                        | non_unary_input_function
                        ;

print_expr_list_opt : /* empty */
                        | print_expr_list
                        ;

print_expr_list  : print_expr
                        | print_expr_list ',' newline_opt print_expr
                        ;

print_expr       : unary_print_expr
                        | non_unary_print_expr
                        ;

unary_print_expr : '+' print_expr
                        | '-' print_expr
                        | unary_print_expr '^'      print_expr
                        | unary_print_expr '*'      print_expr
                        | unary_print_expr '/'      print_expr
                        | unary_print_expr '%'      print_expr
                        | unary_print_expr '+'      print_expr
                        | unary_print_expr '-'      print_expr
                        | unary_print_expr          non_unary_print_expr
                        | unary_print_expr '~'      print_expr
                        | unary_print_expr NO_MATCH print_expr
                        | unary_print_expr In NAME
                        | unary_print_expr AND newline_opt print_expr
                        | unary_print_expr OR  newline_opt print_expr
                        | unary_print_expr '?' print_expr ':' print_expr
                        ;

non_unary_print_expr : '(' expr ')'
                        | '!' print_expr
                        | non_unary_print_expr '^'      print_expr
                        | non_unary_print_expr '*'      print_expr
                        | non_unary_print_expr '/'      print_expr
                        | non_unary_print_expr '%'      print_expr
                        | non_unary_print_expr '+'      print_expr
                        | non_unary_print_expr '-'      print_expr
                        | non_unary_print_expr          non_unary_print_expr
                        | non_unary_print_expr '~'      print_expr
                        | non_unary_print_expr NO_MATCH print_expr
```



```
                        | non_unary_print_expr In NAME
                        | '(' multiple_expr_list ')' In NAME
                        | non_unary_print_expr AND newline_opt print_expr
                        | non_unary_print_expr OR  newline_opt print_expr
                        | non_unary_print_expr '?' print_expr ':' print_expr
                        | NUMBER
                        | STRING
                        | lvalue
                        | ERE
                        | lvalue INCR
                        | lvalue DECR
                        | INCR lvalue
                        | DECR lvalue
                        | lvalue POW_ASSIGN print_expr
                        | lvalue MOD_ASSIGN print_expr
                        | lvalue MUL_ASSIGN print_expr
                        | lvalue DIV_ASSIGN print_expr
                        | lvalue ADD_ASSIGN print_expr
                        | lvalue SUB_ASSIGN print_expr
                        | lvalue '=' print_expr
                        | FUNC_NAME '(' expr_list_opt ')'
                            /* no white space allowed before '(' */
                        | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
                        | BUILTIN_FUNC_NAME
                        ;

lvalue                  : NAME
                        | NAME '[' expr_list ']'
                        | '$' expr
                        ;

non_unary_input_function : simple_get
                        | simple_get '<' expr
                        | non_unary_expr '|' simple_get
                        ;

unary_input_function : unary_expr '|' simple_get
                        ;

simple_get              : GETLINE
                        | GETLINE lvalue
                        ;

newline_opt             : /* empty */
                        | newline_opt NEWLINE
                        ;
```

This grammar has several ambiguities that shall be resolved as follows:

- Operator precedence and associativity shall be as described in Table 3-1 (on page 2608).
- In case of ambiguity, an **else** shall be associated with the most immediately preceding **if** that would satisfy the grammar.

- In some contexts, a <slash> (`'/'`) that is used to surround an ERE could also be the division operator. This shall be resolved in such a way that wherever the division operator could appear, a <slash> is assumed to be the division operator. (There is no unary division operator.)

Each expression in an *awk* program shall conform to the precedence and associativity rules, even when this is not needed to resolve an ambiguity. For example, because `'$'` has higher precedence than `'++'`, the string `"$x++--"` is not a valid *awk* expression, even though it is unambiguously parsed by the grammar as `"$(x++)--"`.

One convention that might not be obvious from the formal grammar is where <newline> characters are acceptable. There are several obvious placements such as terminating a statement, and a <backslash> can be used to escape <newline> characters between any lexical tokens. In addition, <newline> characters without <backslash> characters can follow a comma, an open brace, logical AND operator (`"&&"`), logical OR operator (`"||"`), the **do** keyword, the **else** keyword, and the closing parenthesis of an **if**, **for**, or **while** statement. For example:

```
{ print  $1,
         $2 }
```

Lexical Conventions

The lexical conventions for *awk* programs, with respect to the preceding grammar, shall be as follows:

1. Except as noted, *awk* shall recognize the longest possible token or delimiter beginning at a given point.
2. A comment shall consist of any characters beginning with the <number-sign> character and terminated by, but excluding the next occurrence of, a <newline>. Comments shall have no effect, except to delimit lexical tokens.
3. The <newline> shall be recognized as the token **NEWLINE**.
4. A <backslash> character immediately followed by a <newline> shall have no effect.
5. The token **STRING** shall represent a string constant. A string constant shall begin with the character `'"'`. Within a string constant, a <backslash> character shall be considered to begin an escape sequence as specified in the table in XBD Chapter 5 (on page 113) (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`). In addition, the escape sequences in Table 3-2 (on page 2616) shall be recognized. A <newline> shall not occur within a string constant. A string constant shall be terminated by the first unescaped occurrence of the character `'"'` after the one that begins the string constant. The value of the string shall be the sequence of all unescaped characters and values of escape sequences between, but not including, the two delimiting `'"'` characters.
6. The token **ERE** represents an extended regular expression constant. An ERE constant shall begin with the <slash> character. Within an ERE constant, a <backslash> character shall be considered to begin an escape sequence as specified in the table in XBD Chapter 5 (on page 113). In addition, the escape sequences in Table 3-2 (on page 2616) shall be recognized. The application shall ensure that a <newline> does not occur within an ERE constant. An ERE constant shall be terminated by the first unescaped occurrence of the <slash> character after the one that begins the ERE constant. The extended regular expression represented by the ERE constant shall be the sequence of all unescaped characters and values of escape sequences between, but not including, the two delimiting <slash> characters.

7. A <blank> shall have no effect, except to delimit lexical tokens or within **STRING** or **ERE** tokens.

8. The token **NUMBER** shall represent a numeric constant. Its form and numeric value shall either be equivalent to the **decimal-floating-constant** token as specified by the ISO C standard, or it shall be a sequence of decimal digits and shall be evaluated as an integer constant in decimal. In addition, implementations may accept numeric constants with the form and numeric value equivalent to the **hexadecimal-constant** and **hexadecimal-floating-constant** tokens as specified by the ISO C standard. Note that these forms do not use the radix character from the current locale; they always use a <period>.

   If the value is too large or too small to be representable (see Section 1.1.2, on page 2457), the behavior is undefined.

9. A sequence of underscores, digits, and alphabetics from the portable character set (see XBD Section 6.1, on page 117), beginning with an <underscore> or alphabetic character, shall be considered a word.

10. The following words are keywords that shall be recognized as individual tokens; the name of the token is the same as the keyword:

BEGIN	**delete**	**END**	**function**	**in**	**print**	**while**
break	**do**	**exit**	**getline**	**next**	**printf**	
continue	**else**	**for**	**if**	**nextfile**	**return**	

11. The following words are names of built-in functions and shall be recognized as the token **BUILTIN_FUNC_NAME**:

atan2	**fflush**	**int**	**match**	**split**	**srand**	**system**
close	**gsub**	**length**	**rand**	**sprintf**	**sub**	**tolower**
cos	**index**	**log**	**sin**	**sqrt**	**substr**	**toupper**
exp						

    The above-listed keywords and names of built-in functions are considered reserved words.

12. The token **NAME** shall consist of a word that is not a keyword or a name of a built-in function and is not followed immediately (without any delimiters) by the '(' character.

13. The token **FUNC_NAME** shall consist of a word that is not a keyword or a name of a built-in function, followed immediately (without any delimiters) by the '(' character. The '(' character shall not be included as part of the token.

14. The following two-character sequences shall be recognized as the named tokens:

Token Name	Sequence	Token Name	Sequence
ADD_ASSIGN	+=	**NO_MATCH**	!~
SUB_ASSIGN	−=	**EQ**	==
MUL_ASSIGN	*=	**LE**	<=
DIV_ASSIGN	/=	**GE**	>=
MOD_ASSIGN	%=	**NE**	!=
POW_ASSIGN	^=	**INCR**	++
OR	||	**DECR**	−−
AND	&&	**APPEND**	>>

15. The following single characters shall be recognized as tokens whose names are the character:

    `<newline> { } ( ) [ ] , ; + - * % ^ ! > < | ? : ~ $ =`

There is a lexical ambiguity between the token **ERE** and the tokens `'/'` and **DIV_ASSIGN**. When an input sequence begins with a <slash> character in any syntactic context where the token `'/'` or **DIV_ASSIGN** could appear as the next token in a valid program, the longer of those two tokens that can be recognized shall be recognized. In any other syntactic context where the token **ERE** could appear as the next token in a valid program, the token **ERE** shall be recognized.

EXIT STATUS

The following exit values shall be returned:

0 All input files were processed successfully.

>0 An error occurred.

The exit status can be altered within the program by using an **exit** expression.

CONSEQUENCES OF ERRORS

If any *file* operand is specified and the named file cannot be accessed, *awk* shall write a diagnostic message to standard error and terminate without any further action.

If the program specified by either the *program* operand or a *progfile* operand is not a valid *awk* program (as specified in the EXTENDED DESCRIPTION section), the behavior is undefined.

APPLICATION USAGE

Since <backslash> has a special meaning both in the *assignment* option-argument to the **–v** option and in the *assignment* operand, applications that need to pass strings to *awk* without special interpretation of <backslash> should not use these methods but should instead make use of the **ARGV** or **ENVIRON** array.

The **index**, **length**, **match**, and **substr** functions should not be confused with similar functions in the ISO C standard; the *awk* versions deal with characters, while the ISO C standard deals with bytes.

Because the concatenation operation is represented by adjacent expressions rather than an explicit operator, it is often necessary to use parentheses to enforce the proper evaluation precedence.

When using *awk* to process pathnames, it is recommended that LC_ALL, or at least LC_CTYPE and LC_COLLATE, are set to POSIX or C in the environment, since pathnames can contain byte sequences that do not form valid characters in some locales, in which case the utility's behavior would be undefined. In the POSIX locale each byte is a valid single-byte character, and therefore this problem is avoided.

Since the `"=="` operator checks if strings are identical, not whether they collate equally, applications needing to check whether strings collate equally can use:

```
a <= b && a >= b
```

To specify a *file* operand naming a file with a name containing an <equals-sign>, users can use `"./"` as the first two characters of a relative file pathname that starts with an <underscore> or an alphabetic character to keep the *file* operand from being interpreted as an *assignment* operand. Similarly, `"./-"` can be used to access a file named `'-'` in the current directory rather than use standard input.

EXAMPLES

The *awk* program specified in the command line is most easily specified within single-quotes (for example, *'program'*) for applications using *sh*, because *awk* programs commonly contain characters that are special to the shell, including double-quotes. In the cases where an *awk* program contains single-quote characters, it is usually easiest to specify most of the program as strings within single-quotes concatenated by the shell with quoted single-quote characters. For example:

```
awk '/'\''/ { print "quote:", $0 }'
```

prints all lines from the standard input containing a single-quote character, prefixed with *quote*:.

The following are examples of simple *awk* programs:

1. Write to the standard output all input lines for which field 3 is greater than 5:

   ```
   $3 > 5
   ```

2. Write every tenth line:

   ```
   (NR % 10) == 0
   ```

3. Write any line with a substring matching the regular expression:

   ```
   /(G|D)(2[0-9][[:alpha:]]*)/
   ```

4. Print any line with a substring containing a `'G'` or `'D'`, followed by a sequence of digits and characters. This example uses character classes **digit** and **alpha** to match language-independent digit and alphabetic characters respectively:

   ```
   /(G|D)([[:digit:][:alpha:]]*)/
   ```

5. Write any line in which the second field matches the regular expression and the fourth field does not:

   ```
   $2 ~ /xyz/ && $4 !~ /xyz/
   ```

6. Write any line in which the second field contains a <backslash>:

   ```
   $2 ~ /\\/
   ```

7. Write any line in which the second field contains a <backslash>. Note that <backslash>-escapes are interpreted twice; once in lexical processing of the string and once in processing the regular expression:

   ```
   $2 ~ "\\\\"
   ```

8. Write the second to the last and the last field in each line. Separate the fields by a <colon>:

   ```
   {OFS=":";print $(NF-1), $NF}
   ```

9. Write the line number and number of fields in each line. The three strings representing the line number, the <colon>, and the number of fields are concatenated and that string is written to standard output:

   ```
   {print NR ":" NF}
   ```

10. Write lines longer than 72 characters:

    ```
    length($0) > 72
    ```

11. Write the first two fields in opposite order separated by **OFS**:

    ```
    { print $2, $1 }
    ```


12. Same, with input fields separated by a <comma> or <space> and <tab> characters, or both:

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
      { print $2, $1 }
```

13. Add up the first column, print sum, and average:

```
      {s += $1 }
END   {print "sum is ", s, " average is", s/NR}
```

14. Write fields in reverse order, one per line (many lines out for each line in):

```
{ for (i = NF; i > 0; --i) print $i }
```

15. Write all lines between occurrences of the strings **start** and **stop**:

```
/start/, /stop/
```

16. Write all lines whose first field is different from the previous one:

```
$1 != prev { print; prev = $1 }
```

17. Simulate *echo*:

```
BEGIN  {
        for (i = 1; i < ARGC; ++i)
        printf("%s%s", ARGV[i], i==ARGC-1?"\n":" ")
}
```

18. Write the path prefixes contained in the *PATH* environment variable, one per line:

```
BEGIN  {
        n = split (ENVIRON["PATH"], path, ":")
        for (i = 1; i <= n; ++i)
        print path[i]
}
```

19. If there is a file named **input** containing page headers of the form:

    Page #

    and a file named **program** that contains:

```
/Page/   { $2 = n++; }
         { print }
```

    then the command line:

```
awk -f program n=5 input
```

    prints the file **input**, filling in page numbers starting at 5.

RATIONALE

This description is based on the new *awk*, ``nawk'', (see the referenced *The AWK Programming Language*), which introduced a number of new features to the historical *awk*:

1. New keywords: **delete**, **do**, **function**, **return**
2. New built-in functions: **atan2**, **close**, **cos**, **gsub**, **match**, **rand**, **sin**, **srand**, **sub**, **system**

3. New predefined variables: **FNR**, **ARGC**, **ARGV**, **RSTART**, **RLENGTH**, **SUBSEP**
4. New expression operators: **?**, **:**, ,, ^
5. The **FS** variable and the third argument to **split**, now treated as extended regular expressions.
6. The operator precedence, changed to more closely match the C language. Two examples of code that operate differently are:

```
while ( n /= 10 > 1) ...
if (!"wk" ~ /bwk/) ...
```

Several features have been added based on newer implementations of *awk*:

- Multiple instances of **−f** *progfile* are permitted.
- The new option **−v** *assignment*.
- The new predefined variable **ENVIRON**.
- New built-in functions **toupper** and **tolower**.
- More formatting capabilities are added to **printf** to match the ISO C standard.

Earlier versions of this standard required implementations to support multiple adjacent <semicolon>s, lines with one or more <semicolon> before a rule (*pattern-action* pairs), and lines with only <semicolon>(s). These are not required by this standard and are considered poor programming practice, but can be accepted by an implementation of *awk* as an extension.

The overall *awk* syntax has always been based on the C language, with a few features from the shell command language and other sources. Because of this, it is not completely compatible with any other language, which has caused confusion for some users. It is not the intent of the standard developers to address such issues. A few relatively minor changes toward making the language more compatible with the ISO C standard were made; most of these changes are based on similar changes in recent implementations, as described above. There remain several C-language conventions that are not in *awk*. One of the notable ones is the <comma> operator, which is commonly used to specify multiple expressions in the C language **for** statement. Also, there are various places where *awk* is more restrictive than the C language regarding the type of expression that can be used in a given context. These limitations are due to the different features that the *awk* language does provide.

Regular expressions in *awk* have been extended somewhat from historical implementations to make them a pure superset of extended regular expressions, as defined by POSIX.1-2024 (see XBD Section 9.4, on page 187). The main extensions are internationalization features and interval expressions. Historical implementations of *awk* have long supported <backslash>-escape sequences as an extension to extended regular expressions, and this extension has been retained despite inconsistency with other utilities. The number of escape sequences recognized in both extended regular expressions and strings has varied (generally increasing with time) among implementations. The set specified by POSIX.1-2024 includes most sequences known to be supported by popular implementations and by the ISO C standard. One sequence that is not supported is hexadecimal value escapes beginning with `'\x'`. This would allow values expressed in more than 9 bits to be used within *awk* as in the ISO C standard. However, because this syntax has a non-deterministic length, it does not permit the subsequent character to be a hexadecimal digit. This limitation can be dealt with in the C language by the use of lexical string concatenation. In the *awk* language, concatenation could also be a solution for strings, but not for extended regular expressions (either lexical ERE tokens or strings used dynamically as regular expressions). Because of this limitation, the feature has not been added to

POSIX.1-2024.

When a string variable is used in a context where an extended regular expression normally appears (where the lexical token ERE is used in the grammar) the string does not contain the literal <slash> characters.

Some versions of *awk* allow the form:

```
func name(args, ... ) { statements }
```

This has been deprecated by the authors of the language, who asked that it not be specified.

Historical implementations of *awk* produce an error if a **next** statement is executed in a **BEGIN** action, and cause *awk* to terminate if a **next** statement is executed in an **END** action. This behavior has not been documented, and it was not believed that it was necessary to standardize it.

The specification of conversions between string and numeric values is much more detailed than in the documentation of historical implementations or in the referenced *The AWK Programming Language*. Although most of the behavior is designed to be intuitive, the details are necessary to ensure compatible behavior from different implementations. This is especially important in relational expressions since the types of the operands determine whether a string or numeric comparison is performed. From the perspective of an application developer, it is usually sufficient to expect intuitive behavior and to force conversions (by adding zero or concatenating a null string) when the type of an expression does not obviously match what is needed. The intent has been to specify historical practice in almost all cases. The one exception is that, in historical implementations, variables and constants maintain both string and numeric values after their original value is converted by any use. This means that referencing a variable or constant can have unexpected side-effects. For example, with historical implementations the following program:

```
{
    a = "+2"
    b = 2
    if (NR % 2)
        c = a + b
    if (a == b)
        print "numeric comparison"
    else
        print "string comparison"
}
```

would perform a numeric comparison (and output numeric comparison) for each odd-numbered line, but perform a string comparison (and output string comparison) for each even-numbered line. POSIX.1-2024 ensures that comparisons will be numeric if necessary. With historical implementations, the following program:

```
BEGIN {
    OFMT = "%e"
    print 3.14
    OFMT = "%f"
    print 3.14
}
```

would output `"3.140000e+00"` twice, because in the second **print** statement the constant `"3.14"` would have a string value from the previous conversion. POSIX.1-2024 requires that the output of the second **print** statement be `"3.140000"`. The behavior of historical

implementations was seen as too unintuitive and unpredictable.

It was pointed out that with the rules contained in early drafts, the following script would print nothing:

```
BEGIN {
    y[1.5] = 1
    OFMT = "%e"
    print y[1.5]
}
```

Therefore, a new variable, **CONVFMT**, was introduced. The **OFMT** variable is now restricted to affecting output conversions of numbers to strings and **CONVFMT** is used for internal conversions, such as comparisons or array indexing. The default value is the same as that for **OFMT**, so unless a program changes **CONVFMT** (which no historical program would do), it will receive the historical behavior associated with internal string conversions.

The POSIX *awk* lexical and syntactic conventions are specified more formally than in other sources. Again the intent has been to specify historical practice. One convention that may not be obvious from the formal grammar as in other verbal descriptions is where <newline> characters are acceptable. There are several obvious placements such as terminating a statement, and a <backslash> can be used to escape <newline> characters between any lexical tokens. In addition, <newline> characters without <backslash> characters can follow a comma, an open brace, a logical AND operator (`"&&"`), a logical OR operator (`"||"`), the **do** keyword, the **else** keyword, and the closing parenthesis of an **if**, **for**, or **while** statement. For example:

```
{ print $1,
        $2 }
```

The requirement that *awk* add a trailing <newline> to the program argument text is to simplify the grammar, making it match a text file in form. There is no way for an application or test suite to determine whether a literal <newline> is added or whether *awk* simply acts as if it did.

POSIX.1-2024 requires several changes from historical implementations in order to support internationalization. Probably the most subtle of these is the use of the decimal-point character, defined by the *LC_NUMERIC* category of the locale, in representations of floating-point numbers. This locale-specific character is used in recognizing numeric input, in converting between strings and numeric values, and in formatting output. However, regardless of locale, the <period> character (the decimal-point character of the POSIX locale) is the decimal-point character recognized in processing *awk* programs (including assignments in command line arguments). This is essentially the same convention as the one used in the ISO C standard. The difference is that the C language includes the *setlocale*() function, which permits an application to modify its locale. Because of this capability, a C application begins executing with its locale set to the C locale, and only executes in the environment-specified locale after an explicit call to *setlocale*(). However, adding such an elaborate new feature to the *awk* language was seen as inappropriate for POSIX.1-2024. It is possible to execute an *awk* program explicitly in any desired locale by setting the environment in the shell.

The undefined behavior resulting from NULs in extended regular expressions allows future extensions for the GNU *gawk* program to process binary data.

The behavior in the case of invalid *awk* programs (including lexical, syntactic, and semantic errors) is undefined because it was considered overly limiting on implementations to specify. In most cases such errors can be expected to produce a diagnostic and a non-zero exit status. However, some implementations may choose to extend the language in ways that make use of certain invalid constructs. Other invalid constructs might be deemed worthy of a warning, but

otherwise cause some reasonable behavior. Still other constructs may be very difficult to detect in some implementations. Also, different implementations might detect a given error during an initial parsing of the program (before reading any input files) while others might detect it when executing the program after reading some input. Implementors should be aware that diagnosing errors as early as possible and producing useful diagnostics can ease debugging of applications, and thus make an implementation more usable.

The unspecified behavior from using multi-character **RS** values is to allow possible future extensions based on extended regular expressions used for record separators. Historical implementations take the first character of the string and ignore the others.

Unspecified behavior when *split*(*string*,*array*,<null>) is used is to allow a proposed future extension that would split up a string into an array of individual characters.

In the context of the **getline** function, equally good arguments for different precedences of the **|** and **<** operators can be made. Historical practice has been that:

```
getline < "a" "b"
```

is parsed as:

```
( getline < "a" ) "b"
```

although many would argue that the intent was that the file **ab** should be read. However:

```
getline < "x" + 1
```

parses as:

```
getline < ( "x" + 1 )
```

Similar problems occur with the **|** version of **getline**, particularly in combination with **$**. For example:

```
$"echo hi" | getline
```

(This situation is particularly problematic when used in a **print** statement, where the **|getline** part might be a redirection of the **print**.)

Since in most cases such constructs are not (or at least should not) be used (because they have a natural ambiguity for which there is no conventional parsing), the meaning of these constructs has been made explicitly unspecified. (The effect is that a conforming application that runs into the problem must parenthesize to resolve the ambiguity.) There appeared to be few if any actual uses of such constructs.

Grammars can be written that would cause an error under these circumstances. Where backwards-compatibility is not a large consideration, implementors may wish to use such grammars.

Some historical implementations have allowed some built-in functions to be called without an argument list, the result being a default argument list chosen in some ``reasonable'' way. Use of **length** as a synonym for **length($0)** is the only one of these forms that is thought to be widely known or widely used; this particular form is documented in various places (for example, most historical *awk* reference pages, although not in the referenced *The AWK Programming Language*) as legitimate practice. With this exception, default argument lists have always been undocumented and vaguely defined, and it is not at all clear how (or if) they should be generalized to user-defined functions. They add no useful functionality and preclude possible future extensions that might need to name functions without calling them. Not standardizing them seems the simplest course. The standard developers considered that **length** merited special treatment, however, since it has been documented in the past and sees possibly substantial use in historical

programs. Accordingly, this usage has been made legitimate, but Issue 5 removed the obsolescent marking for XSI-conforming implementations and many otherwise conforming applications depend on this feature.

In **sub** and **gsub**, if *repl* is a string literal (the lexical token **STRING**), then two consecutive <backslash> characters should be used in the string to ensure a single <backslash> will precede the <ampersand> when the resultant string is passed to the function. (For example, to specify one literal <ampersand> in the replacement string, use **gsub**(**ERE**, `"\\&"`).)

Historically, the only special character in the *repl* argument of **sub** and **gsub** string functions was the <ampersand> (`'&'`) character and preceding it with the <backslash> character was used to turn off its special meaning.

The description in the ISO POSIX-2: 1993 standard introduced behavior such that the <backslash> character was another special character and it was unspecified whether there were any other special characters. This description introduced several portability problems, some of which are described below, and so it has been replaced with the more historical description. Some of the problems include:

- Historically, to create the replacement string, a script could use **gsub**(**ERE**, `"\\&"`), but with the ISO POSIX-2: 1993 standard wording, it was necessary to use **gsub**(**ERE**, `"\\\\&"`). The <backslash> characters are doubled here because all string literals are subject to lexical analysis, which would reduce each pair of <backslash> characters to a single <backslash> before being passed to **gsub**.
- Since it was unspecified what the special characters were, for portable scripts to guarantee that characters are printed literally, each character had to be preceded with a <backslash>. (For example, a portable script had to use **gsub**(**ERE**, `"\\h\\i"`) to produce a replacement string of `"hi"`.)

The description for comparisons in the ISO POSIX-2: 1993 standard did not properly describe historical practice because of the way numeric strings are compared as numbers. The current rules cause the following code:

```
if (0 == "000")
    print "strange, but true"
else
    print "not true"
```

to do a numeric comparison, causing the **if** to succeed. It should be intuitively obvious that this is incorrect behavior, and indeed, no historical implementation of *awk* actually behaves this way.

To fix this problem, the definition of *numeric string* was enhanced to include only those values obtained from specific circumstances (mostly external sources) where it is not possible to determine unambiguously whether the value is intended to be a string or a numeric.

Variables that are assigned to a numeric string shall also be treated as a numeric string. (For example, the notion of a numeric string can be propagated across assignments.) In comparisons, all variables having the uninitialized value are to be treated as a numeric operand evaluating to the numeric value zero.

Uninitialized variables include all types of variables including scalars, array elements, and fields. The definition of an uninitialized value in Variables and Special Variables (on page 2612) is necessary to describe the value placed on uninitialized variables and on fields that are valid (for example, **< $NF**) but have no characters in them and to describe how these variables are to be used in comparisons. A valid field, such as **$1**, that has no characters in it can be obtained from an input line of `"\t\t"` when **FS=**`'\t'`. Historically, the comparison (**$1<**10) was done

numerically after evaluating **$1** to the value zero.

The phrase ``... also shall have the numeric value of the numeric string'' was removed from several sections of the ISO POSIX-2:1993 standard because is specifies an unnecessary implementation detail. It is not necessary for POSIX.1-2024 to specify that these objects be assigned two different values. It is only necessary to specify that these objects may evaluate to two different values depending on context.

Historical implementations of *awk* did not parse hexadecimal integer or floating constants like `"0xa"` and `"0xap0"`. Due to an oversight, the 2001 through 2004 editions of this standard required support for hexadecimal floating constants. This was due to the reference to *atof*(). This version of the standard allows but does not require implementations to use *atof*() and includes a description of how floating-point numbers are recognized as an alternative to match historic behavior. The intent of this change is to allow implementations to recognize floating-point constants according to either the ISO/IEC 9899:1990 standard or ISO/IEC 9899:1999 standard, and to allow (but not require) implementations to recognize hexadecimal integer constants.

Historical implementations of *awk* did not support floating-point infinities and NaNs in *numeric strings*; e.g., `"-INF"` and `"NaN"`. However, implementations that use the *atof*() or *strtod*() functions to do the conversion picked up support for these values if they used a ISO/IEC 9899:1999 standard version of the function instead of a ISO/IEC 9899:1990 standard version. Due to an oversight, the 2001 through 2004 editions of this standard did not allow support for infinities and NaNs, but in this revision support is allowed (but not required). This is a silent change to the behavior of *awk* programs; for example, in the POSIX locale the expression:

```
("-INF" + 0 < 0)
```

formerly had the value 0 because `"-INF"` converted to 0, but now it may have the value 0 or 1.

Deleting all elements of an array one element at a time, via:

```
for (index in array)
    delete array[index]
```

is usually not efficient. This standard requires `delete array` to have the same effects, and this was supported in most implementations as a more efficient operation. It is also possible to use `split("", array)` to achieve the same effect and efficiency.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

A future version of this standard may require **srand** to accept any numeric value and calculate the seed by taking the provided value, converting it to an integer, and calculating the integer value modulo 2^n where n is an implementation-defined value greater than or equal to 32.

A future version of this standard may require the initial seed for the **rand** function (the seed value used if **srand** is not called) to be an integer between 0 and 2^n-1 inclusive where n is an implementation-defined value greater than or equal to 32. Additionally, the initial seed value may be required to be a (pseudo-)random value such that two invocations of *awk* are unlikely to emit the same sequence of random values (unless the seed is explicitly set to the same value via **srand**).

A future version of this standard may define a new **posix_srand** function that enables application authors to set the seed to a (pseudo-)random value generated by the system. Alternatively, the specification of the **srand** function may be altered to provide some means to

set the default seed value to a (pseudo-)random value.

SEE ALSO

Section 1.3 (on page 2461), *grep*, *lex*, *sed*

XBD Chapter 5 (on page 113), Section 6.1 (on page 117), Chapter 8 (on page 167), Chapter 9 (on page 179), Section 12.2 (on page 215)

XSH *atof*( ), *exec*, *isspace*( ), *popen*( ), *setlocale*( ), *strtod*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

The *awk* utility is aligned with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

IEEE PASC Interpretation 1003.2 #211 is applied, adding the sentence ``An occurrence of two consecutive <backslash> characters shall be interpreted as just a single literal <backslash> character.'' into the description of the **sub** string function.

Issue 7

PASC Interpretation 1003.2-1992 #107 (SD5-XCU-ERN-73) is applied, updating the description of the **OFS** variable.

Austin Group Interpretation 1003.1-2001 #189 is applied.

Austin Group Interpretation 1003.1-2001 #201 is applied, permitting implementations to support infinities and NaNs.

SD5-XCU-ERN-79 is applied, restoring the horizontal lines to Table 3-1 (on page 2608), and SD5-XCU-ERN-80 is applied, changing the order of some table entries.

SD5-XCU-ERN-87 is applied, updating the descriptive text of the Grammar.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The EXTENDED DESCRIPTION is changed to make the support of hexadecimal integer and floating constants optional.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0057 [224], XCU/TC1-2008/0058 [454], XCU/TC1-2008/0059 [224], XCU/TC1-2008/0060 [224], XCU/TC1-2008/0061 [254], XCU/TC1-2008/0062 [254], XCU/TC1-2008/0063 [224], and XCU/TC1-2008/0064 [454] are applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0058 [584], XCU/TC2-2008/0059 [963], XCU/TC2-2008/0060 [226], XCU/TC2-2008/0061 [663], XCU/TC2-2008/0062 [963], XCU/TC2-2008/0063 [226], and XCU/TC2-2008/0064 [963] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defects 544 and 1136 are applied, requiring implementations to accept the **delete** statement with an unsubscripted array name.

Austin Group Defect 607 is applied, adding the **nextfile** statement.

Austin Group Defect 634 is applied, adding the **fflush** function.

Austin Group Defects 974 and 1451 are applied, clarifying the **ARGC**, **ARGV** and **FILENAME** variables, and adding to APPLICATION USAGE.

Austin Group Defect 983 is applied, changing the descriptions of the **rand** and **srand** functions and the FUTURE DIRECTIONS section.

Austin Group Defect 1070 is applied, requiring the `"!="` and `"=="` operators to perform string comparisons by checking if the strings are identical (and not by checking if they collate equally).

Austin Group Defect 1105 is applied, clarifying the requirements for <backslash> escaping.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1198 is applied, requiring comparisons to be performed numerically when both operands have string values that are numeric strings.

Austin Group Defect 1277 is applied, clarifying that using a <slash> character within an ERE requires escaping only if it is within the lexical token **ERE**.

Austin Group Defect 1320 is applied, clarifying the condition under which ERE matching is against input records.

Austin Group Defect 1395 is applied, changing the requirements for string to number conversion.

Austin Group Defect 1468 is applied, clarifying the behavior when **FS** is an ERE that can match the null string.

Austin Group Defect 1566 is applied, specifying the behavior of the **length** function when passed an array argument.

NAME

basename — return non-directory portion of a pathname

SYNOPSIS

```
basename string [suffix]
```

DESCRIPTION

The *string* operand shall be treated as a pathname, as defined in XBD Section 3.254 (on page 68). The string *string* shall be converted to the filename corresponding to the last pathname component in *string* and then the suffix string *suffix*, if present, shall be removed. This shall be done by performing actions equivalent to the following steps in order:

1. If *string* is a null string, it is unspecified whether the resulting string is `'.'` or a null string. In either case, skip steps 2 through 6.
2. If *string* is `"//"`, it is implementation-defined whether steps 3 to 6 are skipped or processed.
3. If *string* consists entirely of <slash> characters, *string* shall be set to a single <slash> character. In this case, skip steps 4 to 6.
4. If there are any trailing <slash> characters in *string*, they shall be removed.
5. If there are any <slash> characters remaining in *string*, the prefix of *string* up to and including the last <slash> character in *string* shall be removed.
6. If the *suffix* operand is present, is not identical to the characters remaining in *string*, and is identical to a suffix of the characters remaining in *string*, the suffix *suffix* shall be removed from *string*. Otherwise, *string* is not modified by this step. It shall not be considered an error if *suffix* is not found in *string*.

The resulting string shall be written to standard output.

OPTIONS

None.

OPERANDS

The following operands shall be supported:

string A string.

suffix A string.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *basename*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS
Default.

STDOUT
The *basename* utility shall write a line to the standard output in the following format:

```
"%s\n", <resulting string>
```

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
The definition of *pathname* specifies implementation-defined behavior for pathnames starting with two <slash> characters. Therefore, applications shall not arbitrarily add <slash> characters to the beginning of a pathname unless they can ensure that there are more or less than two or are prepared to deal with the implementation-defined consequences.

EXAMPLES
If the string *string* is a valid pathname:

```
$(basename -- "string")
```

produces a filename that could be used to open the file named by *string* in the directory returned by:

```
$(dirname -- "string")
```

If the string *string* is not a valid pathname, the same algorithm is used, but the result need not be a valid filename. The *basename* utility is not expected to make any judgements about the validity of *string* as a pathname; it just follows the specified algorithm to produce a result string.

The following shell script compiles **/usr/src/cmd/cat.c** and moves the output to a file named **cat** in the current directory when invoked with the argument **/usr/src/cmd/cat** or with the argument **/usr/src/cmd/cat.c**:

```
c17 -- "$(dirname -- "$1")/$(basename -- "$1" .c).c" &&
mv a.out "$(basename -- "$1" .c)"
```

The EXAMPLES section of the *basename*() function (see XSH *basename*()) includes a table showing examples of the results of processing several sample pathnames by the *basename*() and

dirname( ) functions and by the *basename* and *dirname* utilities.

RATIONALE

The behaviors of *basename* and *dirname* have been coordinated so that when *string* is a valid pathname:

```
$(basename -- "string")
```

would be a valid filename for the file in the directory:

```
$(dirname -- "string")
```

This would not work for the early proposal versions of these utilities due to the way it specified handling of trailing <slash> characters.

Since the definition of *pathname* specifies implementation-defined behavior for pathnames starting with two <slash> characters, this volume of POSIX.1-2024 specifies similar implementation-defined behavior for the *basename* and *dirname* utilities.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

Section 2.5 (on page 2478), *dirname*

XBD Section 3.254 (on page 68), Chapter 8 (on page 167)

XSH *basename*( ), *dirname*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

IEEE PASC Interpretation 1003.2 #164 is applied.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0065 [192,538], XCU/TC1-2008/0066 [192,538], and XCU/TC1-2008/0067 [192,430,538] are applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0065 [612] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

batch — schedule commands to be executed in a batch queue

SYNOPSIS

```
batch
```

DESCRIPTION

The *batch* utility shall read commands from standard input and schedule them for execution in a batch queue. It shall be the equivalent of the command:

```
at -q b -m now
```

where queue *b* is a special *at* queue, specifically for batch jobs. Batch jobs shall be submitted to the batch queue with no time constraints and shall be run by the system using algorithms, based on unspecified factors, that may vary with each invocation of *batch*.

XSI Users shall be permitted to use *batch* if their name appears in the file **at.allow** which is located in an implementation-defined directory. If that file does not exist, the file **at.deny**, which is located in an implementation-defined directory, shall be checked to determine whether the user shall be denied access to *batch*. If neither file exists, only a process with appropriate privileges shall be allowed to submit a job. If only **at.deny** exists and is empty, global usage shall be permitted. The **at.allow** and **at.deny** files shall consist of one user name per line.

OPTIONS

None.

OPERANDS

None.

STDIN

The standard input shall be a text file consisting of commands acceptable to the shell command language described in Chapter 2 (on page 2472).

INPUT FILES

XSI The text files **at.allow** and **at.deny**, which are located in an implementation-defined directory, shall contain zero or more user names, one per line, of users who are, respectively, authorized or denied access to the *at* and *batch* utilities.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *batch*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

LC_TIME Determine the format and contents for date and time strings written by *batch*.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

SHELL Determine the name of a command interpreter to be used to invoke the at-job. If the variable is unset or null, *sh* shall be used. If it is set to a value other than a name for *sh*, the implementation shall do one of the following: use that shell; use *sh*; use the login shell from the user database; any of the preceding accompanied by a warning diagnostic about which was chosen.

TZ Determine the timezone. The job shall be submitted for execution at the time specified by *timespec* or **–t** *time* relative to the timezone specified by the *TZ* variable. If *timespec* specifies a timezone, it overrides *TZ*. If *timespec* does not specify a timezone and *TZ* is unset or null, an unspecified default timezone shall be used.

ASYNCHRONOUS EVENTS

Default.

STDOUT

When standard input is a terminal, prompts of unspecified format for each line of the user input described in the STDIN section may be written to standard output.

STDERR

The following shall be written to standard error when a job has been successfully submitted:

```
"job %s at %s\n", at_job_id, <date>
```

where *date* shall be equivalent in format to the output of:

```
date +"%a %b %e %T %Y"
```

The date and time written shall be adjusted so that they appear in the timezone of the user (as determined by the *TZ* variable).

Neither this, nor warning messages concerning the selection of the command interpreter, are considered a diagnostic that changes the exit status.

Diagnostic messages, if any, shall be written to standard error.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

The job shall not be scheduled.

APPLICATION USAGE

It may be useful to redirect standard output within the specified commands.

EXAMPLES

1. This sequence can be used at a terminal:

```
batch
sort < file >outfile
EOT
```

2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a command procedure (the sequence of output redirection specifications is significant):

```
batch <<!
diff file1 file2 2>&1 >outfile | mailx -s "outfile update" mygroup
!
```

   Note that this always sends mail when there has been an attempt to update **outfile** and the body of the message will be empty unless an error occurred.

3. The following shows how to capture both standard error and standard output:

```
batch <<EOF
{
    run-batch-processing |
        mailx -s "batch processing output" mygroup
} 2>&1 | mailx -E -s "errors during batch processing" mygroup
EOF
```

RATIONALE

Early proposals described *batch* in a manner totally separated from *at*, even though the historical model treated it almost as a synonym for *at* **−qb**. A number of features were added to list and control batch work separately from those in *at*. Upon further reflection, it was decided that the benefit of this did not merit the change to the historical interface.

The **−m** option was included on the equivalent *at* command because it is historical practice to mail results to the submitter, even if all job-produced output is redirected. As explained in the RATIONALE for *at*, the **now** keyword submits the job for immediate execution (after scheduling delays), despite some historical systems where *at* **now** would have been considered an error.

FUTURE DIRECTIONS

None.

SEE ALSO

at

XBD Chapter 8 (on page 167)

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The NAME is changed to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

87067 **Issue 7**

87068 The *batch* utility is moved from the User Portability Utilities option to the Base. User Portability
87069 Utilities is now an option for interactive utilities.

87070 SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced
87071 by the *batch* utility.

87072 **Issue 8**

87073 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

87074 Austin Group Defect 1368 is applied, changing the EXAMPLES section.

NAME

bc — arbitrary-precision arithmetic language

SYNOPSIS

```
bc [-l] [file...]
```

DESCRIPTION

The *bc* utility shall implement an arbitrary precision calculator. It shall take input from any files given, then read from the standard input. If the standard input and standard output to *bc* are attached to a terminal, the invocation of *bc* shall be considered to be *interactive*, causing behavioral constraints described in the following sections.

OPTIONS

The *bc* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

–l (The letter ell.) Define the math functions and initialize *scale* to 20, instead of the default zero; see the EXTENDED DESCRIPTION section.

OPERANDS

The following operand shall be supported:

file A pathname of a text file containing *bc* program statements. After all *file*s have been read, *bc* shall read the standard input.

STDIN

See the INPUT FILES section.

INPUT FILES

Input files shall be text files containing a sequence of comments, statements, and function definitions that shall be executed as they are read.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *bc*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The output of the *bc* utility shall be controlled by the program read, and consist of zero or more lines containing the value of all executed expressions without assignments. The radix and precision of the output shall be controlled by the values of the **obase** and **scale** variables; see the EXTENDED DESCRIPTION section.

87119 **STDERR**

87120 The standard error shall be used only for diagnostic messages.

87121 **OUTPUT FILES**

87122 None.

87123 **EXTENDED DESCRIPTION**

87124 **Grammar**

87125 The grammar in this section and the lexical conventions in the following section shall together
87126 describe the syntax for *bc* programs. The general conventions for this style of grammar are
87127 described in Section 1.3 (on page 2461). A valid program can be represented as the non-terminal
87128 symbol **program** in the grammar. This formal syntax shall take precedence over the text syntax
87129 description.

```
87130 %token    EOF NEWLINE STRING LETTER NUMBER

87131 %token    MUL_OP
87132 /*        '*', '/', '%'                              */

87133 %token    ASSIGN_OP
87134 /*        '=', '+=', '-=', '*=', '/=', '%=', '^=' */

87135 %token    REL_OP
87136 /*        '==', '<=', '>=', '!=', '<', '>'        */

87137 %token    INCR_DECR
87138 /*        '++', '--'                                 */

87139 %token    Define    Break    Quit    Length
87140 /*        'define', 'break', 'quit', 'length'     */

87141 %token    Return    For    If    While    Sqrt
87142 /*        'return', 'for', 'if', 'while', 'sqrt'  */

87143 %token    Scale    Ibase    Obase    Auto
87144 /*        'scale', 'ibase', 'obase', 'auto'       */

87145 %start    program

87146 %%

87147 program              : EOF
87148                      | input_item program
87149                      ;

87150 input_item           : semicolon_list NEWLINE
87151                      | function
87152                      ;

87153 semicolon_list       : /* empty */
87154                      | statement
87155                      | semicolon_list ';' statement
87156                      | semicolon_list ';'
87157                      ;

87158 statement_list       : /* empty */
87159                      | statement
87160                      | statement_list NEWLINE
```



```
                            | statement_list NEWLINE statement
                            | statement_list ';'
                            | statement_list ';' statement
                            ;

    statement               : expression
                            | STRING
                            | Break
                            | Quit
                            | Return
                            | Return '(' return_expression ')'
                            | For '(' expression ';'
                                relational_expression ';'
                                expression ')' statement
                            | If '(' relational_expression ')' statement
                            | While '(' relational_expression ')' statement
                            | '{' statement_list '}'
                            ;

    function                : Define LETTER '(' opt_define_list ')'
                                '{' NEWLINE opt_auto_define_list
                                statement_list '}'
                            ;

    opt_define_list         : /* empty */
                            | define_list
                            ;

    opt_auto_define_list    : /* empty */
                            | Auto define_list NEWLINE
                            | Auto define_list ';'
                            ;

    define_list             : LETTER
                            | LETTER '[' ']'
                            | define_list ',' LETTER
                            | define_list ',' LETTER '[' ']'
                            ;

    opt_argument_list       : /* empty */
                            | argument_list
                            ;

    argument_list           : expression
                            | expression ',' argument_list
                            | LETTER '[' ']'
                            | LETTER '[' ']' ',' argument_list
                            ;

    relational_expression   : expression
                            | expression REL_OP expression
                            ;

    return_expression       : /* empty */
                            | expression
                            ;
```



```
expression          : named_expression
                    | NUMBER
                    | '(' expression ')'
                    | LETTER '(' opt_argument_list ')'
                    | '-' expression
                    | expression '+' expression
                    | expression '-' expression
                    | expression MUL_OP expression
                    | expression '^' expression
                    | INCR_DECR named_expression
                    | named_expression INCR_DECR
                    | named_expression ASSIGN_OP expression
                    | Length '(' expression ')'
                    | Sqrt '(' expression ')'
                    | Scale '(' expression ')'
                    ;
named_expression    : LETTER
                    | LETTER '[' expression ']'
                    | Scale
                    | Ibase
                    | Obase
                    ;
```

Lexical Conventions in bc

The lexical conventions for *bc* programs, with respect to the preceding grammar, shall be as follows:

1. Except as noted, *bc* shall recognize the longest possible token or delimiter beginning at a given point.
2. A comment shall consist of any characters beginning with the two adjacent characters `"/*"` and terminated by the next occurrence of the two adjacent characters `"*/"`. Comments shall have no effect except to delimit lexical tokens.
3. The <newline> shall be recognized as the token **NEWLINE**.
4. The token **STRING** shall represent a string constant; it shall consist of any characters beginning with the double-quote character (`'"'`) and terminated by another occurrence of the double-quote character. The value of the string is the sequence of all characters between, but not including, the two double-quote characters. All characters shall be taken literally from the input, and there is no way to specify a string containing a double-quote character. The length of the value of each string shall be limited to {BC_STRING_MAX} bytes.
5. A <blank> shall have no effect except as an ordinary character if it appears within a **STRING** token, or to delimit a lexical token other than **STRING**.
6. The combination of a <backslash> character immediately followed by a <newline> shall have no effect other than to delimit lexical tokens with the following exceptions:
   - It shall be interpreted as the character sequence `"\<newline>"` in **STRING** tokens.

87251 • It shall be ignored as part of a multi-line **NUMBER** token.

87252 7. The token **NUMBER** shall represent a numeric constant. It shall be recognized by the
87253 following grammar:

```
87254 NUMBER  : integer
87255         | '.' integer
87256         | integer '.'
87257         | integer '.' integer
87258         ;

87259 integer : digit
87260         | integer digit
87261         ;

87262 digit   : 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
87263         | 8 | 9 | A | B | C | D | E | F
87264         ;
```

87265 8. The value of a **NUMBER** token shall be interpreted as a numeral in the base specified by
87266 the value of the internal register **ibase** (described below). Each of the **digit** characters
87267 shall have the value from 0 to 15 in the order listed here, and the <period> character shall
87268 represent the radix point. The behavior is undefined if digits greater than or equal to the
87269 value of **ibase** appear in the token. However, note the exception for single-digit values
87270 being assigned to **ibase** and **obase** themselves, in Operations in bc (on page 2656).

87271 9. The following keywords shall be recognized as tokens:

87272 **auto** **ibase** **length** **return** **while**
87273 **break** **if** **obase** **scale**
87274 **define** **for** **quit** **sqrt**

87275 10. Any of the following characters occurring anywhere except within a keyword shall be
87276 recognized as the token **LETTER**:

87277 `a b c d e f g h i j k l m n o p q r s t u v w x y z`

87278 11. The following single-character and two-character sequences shall be recognized as the
87279 token **ASSIGN_OP**:

87280 `=  +=  -=  *=  /=  %=  ^=`

87281 12. If an `'='` character, as the beginning of a token, is followed by a `'-'` character with no
87282 intervening delimiter, the behavior is undefined.

87283 13. The following single-characters shall be recognized as the token **MUL_OP**:

87284 `*  /  %`

87285 14. The following single-character and two-character sequences shall be recognized as the
87286 token **REL_OP**:

87287 `==  <=  >=  !=  <  >`

87288 15. The following two-character sequences shall be recognized as the token **INCR_DECR**:

87289 `++  --`

87290 16. The following single characters shall be recognized as tokens whose names are the
87291 character:

87292 `<newline>  (  )  ,  +  -  ;  [  ]  ^  {  }`

87293 17. The token **EOF** is returned when the end of input is reached.

87294 **Operations in bc**

87295 There are three kinds of identifiers: ordinary identifiers, array identifiers, and function
87296 identifiers. All three types consist of single lowercase letters. Array identifiers shall be followed
87297 by square brackets (`"[]"`). An array subscript is required except in an argument or auto list.
87298 Arrays are singly dimensioned and can contain up to {BC_DIM_MAX} elements. Indexing shall
87299 begin at zero so an array is indexed from 0 to {BC_DIM_MAX}–1. Subscripts shall be truncated
87300 to integers. The application shall ensure that function identifiers are followed by parentheses,
87301 possibly enclosing arguments. The three types of identifiers do not conflict.

87302 The following table summarizes the rules for precedence and associativity of all operators.
87303 Operators on the same line shall have the same precedence; rows are in order of decreasing
87304 precedence.

87305 **Table 3-3** Operators in *bc*

87306

Operator	Associativity
`++, --`	N/A
`unary -`	N/A
`^`	Right to left
`*, /, %`	Left to right
`+, binary -`	Left to right
`=, +=, -=, *=, /=, %=, ^=`	Right to left
`==, <=, >=, !=, <, >`	None

87313

87314 Each expression or named expression has a *scale*, which is the number of decimal digits that
87315 shall be maintained as the fractional portion of the expression.

87316 *Named expressions* are places where values are stored. Named expressions shall be valid on the
87317 left side of an assignment. The value of a named expression shall be the value stored in the place
87318 named. Simple identifiers and array elements are named expressions; they have an initial value
87319 of zero and an initial scale of zero.

87320 The internal registers **scale**, **ibase**, and **obase** are all named expressions. The scale of an
87321 expression consisting of the name of one of these registers shall be zero; values assigned to any
87322 of these registers are truncated to integers. The **scale** register shall contain a global value used in
87323 computing the scale of expressions (as described below). The value of the register **scale** is
87324 limited to 0 ≤ **scale** ≤ {BC_SCALE_MAX} and shall have a default value of zero. The **ibase** and
87325 **obase** registers are the input and output number radix, respectively. The value of **ibase** shall be
87326 limited to:

87327 `2 ≤ ibase ≤ 16`

87328 The value of **obase** shall be limited to:

87329 `2 ≤ obase ≤ {BC_BASE_MAX}`

87330 When either **ibase** or **obase** is assigned a single **digit** value from the list in Lexical Conventions
87331 in bc (on page 2654), the value shall be assumed in hexadecimal. (For example, **ibase**=A sets to
87332 base ten, regardless of the current **ibase** value.) Otherwise, the behavior is undefined when
87333 digits greater than or equal to the value of **ibase** appear in the input. Both **ibase** and **obase** shall
87334 have initial values of 10.

Internal computations shall be conducted as if in decimal, regardless of the input and output bases, to the specified number of decimal digits. When an exact result is not achieved (for example, **scale**=0; 3.2/1)**,** the result shall be truncated.

For all values of **obase** specified by this volume of POSIX.1-2024, *bc* shall output numeric values by performing each of the following steps in order:

1. If the value is less than zero, a <hyphen-minus> (`'-'`) character shall be output.
2. One of the following is output, depending on the numerical value:
   - If the absolute value of the numerical value is greater than or equal to one, the integer portion of the value shall be output as a series of digits appropriate to **obase** (as described below), most significant digit first. The most significant non-zero digit shall be output next, followed by each successively less significant digit.
   - If the absolute value of the numerical value is less than one but greater than zero and the scale of the numerical value is greater than zero, it is unspecified whether the character 0 is output.
   - If the numerical value is zero, the character 0 shall be output.
3. If the scale of the value is greater than zero and the numeric value is not zero, a <period> character shall be output, followed by a series of digits appropriate to **obase** (as described below) representing the most significant portion of the fractional part of the value. If *s* represents the scale of the value being output, the number of digits output shall be *s* if **obase** is 10, less than or equal to *s* if **obase** is greater than 10, or greater than or equal to *s* if **obase** is less than 10. For **obase** values other than 10, this should be the number of digits needed to represent a precision of 10^s.

For **obase** values from 2 to 16, valid digits are the first **obase** of the single characters:

```
0 1 2 3 4 5 6 7 8 9 A B C D E F
```

which represent the values zero to 15, inclusive, respectively.

For bases greater than 16, each digit shall be written as a separate multi-digit decimal number. Each digit except the most significant fractional digit shall be preceded by a single <space>. For bases from 17 to 100, *bc* shall write two-digit decimal numbers; for bases from 101 to 1 000, three-digit decimal strings, and so on. For example, the decimal number 1 024 in base 25 would be written as:

```
Δ01Δ15Δ24
```

and in base 125, as:

```
Δ008Δ024
```

Very large numbers shall be split across lines with 70 characters per line in the POSIX locale; other locales may split at different character boundaries. Lines that are continued shall end with a <backslash>.

A function call shall consist of a function name followed by parentheses containing a <comma>-separated list of expressions, which are the function arguments. A whole array passed as an argument shall be specified by the array name followed by empty square brackets. All function arguments shall be passed by value. As a result, changes made to the formal parameters shall have no effect on the actual arguments. If the function terminates by executing a **return** statement, the value of the function shall be the value of the expression in the parentheses of the **return** statement or shall be zero if no expression is provided or if there is no **return** statement.

The result of **sqrt**(*expression*) shall be the square root of the expression. The result shall be truncated in the least significant decimal place. The scale of the result shall be the scale of the expression or the value of **scale**, whichever is larger.

The result of **length**(*expression*) shall be the total number of significant decimal digits in the expression. The scale of the result shall be zero.

The result of **scale**(*expression*) shall be the scale of the expression. The scale of the result shall be zero.

A numeric constant shall be an expression. The scale shall be the number of digits that follow the radix point in the input representing the constant, or zero if no radix point appears.

The sequence (*expression*) shall be an expression with the same value and scale as *expression*. The parentheses can be used to alter the normal precedence.

The semantics of the unary and binary operators are as follows:

–expression
: The result shall be the negative of the *expression*. The scale of the result shall be the scale of *expression*.

The unary increment and decrement operators shall not modify the scale of the named expression upon which they operate. The scale of the result shall be the scale of that named expression.

++named-expression
: The named expression shall be incremented by one. The result shall be the value of the named expression after incrementing.

––named-expression
: The named expression shall be decremented by one. The result shall be the value of the named expression after decrementing.

named-expression++
: The named expression shall be incremented by one. The result shall be the value of the named expression before incrementing.

named-expression––
: The named expression shall be decremented by one. The result shall be the value of the named expression before decrementing.

The exponentiation operator, <circumflex> (`'^'`), shall bind right to left.

expression^expression
: The result shall be the first *expression* raised to the power of the second *expression*. If the second expression is not an integer, the behavior is undefined. If *a* is the scale of the left expression and *b* is the absolute value of the right expression, the scale of the result shall be:

```
if b >= 0 min(a * b, max(scale, a)) if b < 0 scale
```

The multiplicative operators (`'*'`, `'/'`, `'%'`) shall bind left to right.

*expression*expression*
: The result shall be the product of the two expressions. If *a* and *b* are the scales of the two expressions, then the scale of the result shall be:

```
min(a+b,max(scale,a,b))
```


87420 *expression*/*expression*
87421 The result shall be the quotient of the two expressions. The scale of the result shall be the
87422 value of **scale**.

87423 *expression*%*expression*
87424 For expressions *a* and *b*, *a*%*b* shall be evaluated equivalent to the steps:

87425 1. Compute *a*/*b* to current scale.

87426 2. Use the result to compute:

87427 `a - (a / b) * b`

87428 to scale:

87429 `max(scale + scale(b), scale(a))`

87430 The scale of the result shall be:

87431 `max(scale + scale(b), scale(a))`

87432 When **scale** is zero, the `'%'` operator is the mathematical remainder operator.

87433 The additive operators (`'+'`, `'−'`) shall bind left to right.

87434 *expression*+*expression*
87435 The result shall be the sum of the two expressions. The scale of the result shall be the
87436 maximum of the scales of the expressions.

87437 *expression*–*expression*
87438 The result shall be the difference of the two expressions. The scale of the result shall be the
87439 maximum of the scales of the expressions.

87440 The assignment operators (`'='`, `"+="`, `"−="`, `"*="`, `"/="`, `"%="`, `"^="`) shall bind right to left.

87441 *named-expression*=*expression*
87442 This expression shall result in assigning the value of the expression on the right to the
87443 named expression on the left. The scale of both the named expression and the result shall be
87444 the scale of *expression*.

87445 The compound assignment forms:

87446 *named−expression <operator>= expression*

87447 shall be equivalent to:

87448 *named−expression=named−expression <operator> expression*

87449 except that the *named-expression* shall be evaluated only once.

87450 Unlike all other operators, the relational operators (`'<'`, `'>'`, `"<="`, `">="`, `"=="`, `"!="`) shall be
87451 only valid as the object of an **if**, **while**, or inside a **for** statement.

87452 *expression1*<*expression2*
87453 The relation shall be true if the value of *expression1* is strictly less than the value of
87454 *expression2*.

87455 *expression1*>*expression2*
87456 The relation shall be true if the value of *expression1* is strictly greater than the value of
87457 *expression2*.

87458 *expression1*<=*expression2*
87459 The relation shall be true if the value of *expression1* is less than or equal to the value of
87460 *expression2*.

87461 *expression1*>=*expression2*
87462 The relation shall be true if the value of *expression1* is greater than or equal to the value of
87463 *expression2*.

87464 *expression1*==*expression2*
87465 The relation shall be true if the values of *expression1* and *expression2* are equal.

87466 *expression1*!=*expression2*
87467 The relation shall be true if the values of *expression1* and *expression2* are unequal.

87468 There are only two storage classes in *bc*: global and automatic (local). Only identifiers that are
87469 local to a function need be declared with the **auto** command. The arguments to a function shall
87470 be local to the function. All other identifiers are assumed to be global and available to all
87471 functions. All identifiers, global and local, have initial values of zero. Identifiers declared as auto
87472 shall be allocated on entry to the function and released on returning from the function. They
87473 therefore do not retain values between function calls. Auto arrays shall be specified by the array
87474 name followed by empty square brackets. On entry to a function, the old values of the names
87475 that appear as parameters and as automatic variables shall be pushed onto a stack. Until the
87476 function returns, reference to these names shall refer only to the new values.

87477 References to any of these names from other functions that are called from this function also
87478 refer to the new value until one of those functions uses the same name for a local variable.

87479 When a statement is an expression, unless the main operator is an assignment, execution of the
87480 statement shall write the value of the expression followed by a <newline>.

87481 When a statement is a string, execution of the statement shall write the value of the string.

87482 Statements separated by <semicolon> or <newline> characters shall be executed sequentially. In
87483 an interactive invocation of *bc*, each time a <newline> is read that satisfies the grammatical
87484 production:

87485 `input_item : semicolon_list NEWLINE`

87486 the sequential list of statements making up the **semicolon_list** shall be executed immediately
87487 and any output produced by that execution shall be written without any delay due to buffering.

87488 In an **if** statement (**if**(*relation*) *statement*), the *statement* shall be executed if the relation is true.

87489 The **while** statement (**while**(*relation*) *statement*) implements a loop in which the *relation* is tested;
87490 each time the *relation* is true, the *statement* shall be executed and the *relation* retested. When the
87491 *relation* is false, execution shall resume after *statement*.

87492 A **for** statement(**for**(*expression*; *relation*; *expression*) *statement*) shall be the same as:

```
87493 first-expression
87494 while (relation) {
87495     statement
87496     last-expression
87497 }
```

87498 The application shall ensure that all three expressions are present.

87499 The **break** statement shall cause termination of a **for** or **while** statement.

87500 The **auto** statement (**auto** *identifier* **[**,*identifier***]** ...) shall cause the values of the identifiers to be

pushed down. The identifiers can be ordinary identifiers or array identifiers. Array identifiers shall be specified by following the array name by empty square brackets. The application shall ensure that the **auto** statement is the first statement in a function definition.

A **define** statement:

```
define LETTER ( opt_define_list ) {
    opt_auto_define_list
    statement_list
}
```

defines a function named **LETTER**. If a function named **LETTER** was previously defined, the **define** statement shall replace the previous definition. The expression:

```
LETTER ( opt_argument_list )
```

shall invoke the function named **LETTER**. The behavior is undefined if the number of arguments in the invocation does not match the number of parameters in the definition. Functions shall be defined before they are invoked. A function shall be considered to be defined within its own body, so recursive calls are valid. The values of numeric constants within a function shall be interpreted in the base specified by the value of the **ibase** register when the function is invoked.

The **return** statements (**return** and **return**(*expression*)) shall cause termination of a function, popping of its auto variables, and specification of the result of the function. The first form shall be equivalent to **return**(0). The value and scale of the result returned by the function shall be the value and scale of the expression returned.

The **quit** statement (**quit**) shall stop execution of a *bc* program at the point where the statement occurs in the input, even if it occurs in a function definition, or in an **if**, **for**, or **while** statement.

The following functions shall be defined when the **–l** option is specified:

s(*expression*)
: Sine of argument in radians.

c(*expression*)
: Cosine of argument in radians.

a(*expression*)
: Arctangent of argument.

l(*expression*)
: Natural logarithm of argument.

e(*expression*)
: Exponential function of argument.

j(*expression1*, *expression2*)
: Bessel function of *expression2* of the first kind of integer order *expression1*.

The scale of the result returned by these functions shall be the value of the **scale** register at the time the function is invoked. The value of the **scale** register after these functions have completed their execution shall be the same value it had upon invocation. The behavior is undefined if any of these functions is invoked with an argument outside the domain of the mathematical function.

EXIT STATUS

The following exit values shall be returned:

0 All input files were processed successfully.

unspecified An error occurred.

CONSEQUENCES OF ERRORS

If any *file* operand is specified and the named file cannot be accessed, *bc* shall write a diagnostic message to standard error and terminate without any further action.

In an interactive invocation of *bc*, the utility should print an error message and recover following any error in the input. In a non-interactive invocation of *bc*, invalid input causes undefined behavior.

APPLICATION USAGE

Automatic variables in *bc* do not work in exactly the same way as in either C or PL/1.

For historical reasons, the exit status from *bc* cannot be relied upon to indicate that an error has occurred. Returning zero after an error is possible. Therefore, *bc* should be used primarily by interactive users (who can react to error messages) or by application programs that can somehow validate the answers returned as not including error messages.

The *bc* utility always uses the <period> (`'.'`) character to represent a radix point, regardless of any decimal-point character specified as part of the current locale. In languages like C or *awk*, the <period> character is used in program source, so it can be portable and unambiguous, while the locale-specific character is used in input and output. Because there is no distinction between source and input in *bc*, this arrangement would not be possible. Using the locale-specific character in *bc*'s input would introduce ambiguities into the language; consider the following example in a locale with a <comma> as the decimal-point character:

```
define f(a,b) {
    ...
}
...
f(1,2,3)
```

Because of such ambiguities, the <period> character is used in input. Having input follow different conventions from output would be confusing in either pipeline usage or interactive usage, so the <period> is also used in output.

EXAMPLES

In the shell, the following assigns an approximation of the first ten digits of `'π'` to the variable *x*:

```
x=$(printf "%s\n" 'scale = 10; 104348/33215' | bc)
```

The following *bc* program prints the same approximation of `'π'`, with a label, to standard output:

```
scale = 10
"pi equals "
104348 / 33215
```

The following defines a function to compute an approximate value of the exponential function (note that such a function is predefined if the **–l** option is specified):

```
scale = 20
define e(x){
    auto a, b, c, i, s
```



```
        a = 1
        b = 1
        s = 1
        for (i = 1; 1 == 1; i++){
            a = a*x
            b = b*i
            c = a/b
            if (c == 0) {
                 return(s)
            }
            s = s+c
        }
    }
```

The following prints approximate values of the exponential function of the first ten integers:

```
    for (i = 1; i <= 10; ++i) {
        e(i)
    }
```

RATIONALE

The *bc* utility is implemented historically as a front-end processor for *dc*; *dc* was not selected to be part of this volume of POSIX.1-2024 because *bc* was thought to have a more intuitive programmatic interface. Current implementations that implement *bc* using *dc* are expected to be compliant.

The exit status for error conditions has been left unspecified for several reasons:

- The *bc* utility is used in both interactive and non-interactive situations. Different exit codes may be appropriate for the two uses.
- It is unclear when a non-zero exit should be given; divide-by-zero, undefined functions, and syntax errors are all possibilities.
- It is not clear what utility the exit status has.
- In the 4.3 BSD, System V, and Ninth Edition implementations, *bc* works in conjunction with *dc*. The *dc* utility is the parent, *bc* is the child. This was done to cleanly terminate *bc* if *dc* aborted.

The decision to have *bc* exit upon encountering an inaccessible input file is based on the belief that *bc file1 file2* is used most often when at least *file1* contains data/function declarations/initializations. Having *bc* continue with prerequisite files missing is probably not useful. There is no implication in the CONSEQUENCES OF ERRORS section that *bc* must check all its files for accessibility before opening any of them.

There was considerable debate on the appropriateness of the language accepted by *bc*. Several reviewers preferred to see either a pure subset of the C language or some changes to make the language more compatible with C. While the *bc* language has some obvious similarities to C, it has never claimed to be compatible with any version of C. An interpreter for a subset of C might be a very worthwhile utility, and it could potentially make *bc* obsolete. However, no such utility is known in historical practice, and it was not within the scope of this volume of POSIX.1-2024 to define such a language and utility. If and when they are defined, it may be appropriate to include them in a future version of this standard. This left the following alternatives:

1. Exclude any calculator language from this volume of POSIX.1-2024.

   The consensus of the standard developers was that a simple programmatic calculator language is very useful for both applications and interactive users. The only arguments for excluding any calculator were that it would become obsolete if and when a C-compatible one emerged, or that the absence would encourage the development of such a C-compatible one. These arguments did not sufficiently address the needs of current application developers.

2. Standardize the historical *dc*, possibly with minor modifications.

   The consensus of the standard developers was that *dc* is a fundamentally less usable language and that that would be far too severe a penalty for avoiding the issue of being similar to but incompatible with C.

3. Standardize the historical *bc*, possibly with minor modifications.

   This was the approach taken. Most of the proponents of changing the language would not have been satisfied until most or all of the incompatibilities with C were resolved. Since most of the changes considered most desirable would break historical applications and require significant modification to historical implementations, almost no modifications were made. The one significant modification that was made was the replacement of the historical *bc* assignment operators `"=+"`, and so on, with the more modern `"+="`, and so on. The older versions are considered to be fundamentally flawed because of the lexical ambiguity in uses like *a*=–1.

   In order to permit implementations to deal with backwards-compatibility as they see fit, the behavior of this one ambiguous construct was made undefined. (At least three implementations have been known to support this change already, so the degree of change involved should not be great.)

The `'%'` operator is the mathematical remainder operator when **scale** is zero. The behavior of this operator for other values of **scale** is from historical implementations of *bc*, and has been maintained for the sake of historical applications despite its non-intuitive nature.

Historical implementations permit setting **ibase** and **obase** to a broader range of values. This includes values less than 2, which were not seen as sufficiently useful to standardize. These implementations do not interpret input properly for values of **ibase** that are greater than 16. This is because numeric constants are recognized syntactically, rather than lexically, as described in this volume of POSIX.1-2024. They are built from lexical tokens of single hexadecimal digits and <period> characters. Since <blank> characters between tokens are not visible at the syntactic level, it is not possible to recognize the multi-digit ``digits'' used in the higher bases properly. The ability to recognize input in these bases was not considered useful enough to require modifying these implementations. Note that the recognition of numeric constants at the syntactic level is not a problem with conformance to this volume of POSIX.1-2024, as it does not impact the behavior of conforming applications (and correct *bc* programs). Historical implementations also accept input with all of the digits `'0'–'9'` and `'A'–'F'` regardless of the value of **ibase**; since digits with value greater than or equal to **ibase** are not really appropriate, the behavior when they appear is undefined, except for the common case of:

```
ibase=8;
    /* Process in octal base. */
...
ibase=A
    /* Restore decimal base. */
```

In some historical implementations, if the expression to be written is an uninitialized array

element, a leading <space> and/or up to four leading 0 characters may be output before the character zero. This behavior is considered a bug; it is unlikely that any currently conforming application relies on:

```
echo 'b[3]' | bc
```

returning 00000 rather than 0.

Exact calculation of the number of fractional digits to output for a given value in a base other than 10 can be computationally expensive. Historical implementations use a faster approximation, and this is permitted. Note that the requirements apply only to values of **obase** that this volume of POSIX.1-2024 requires implementations to support (in particular, not to 1, 0, or negative bases, if an implementation supports them as an extension).

Historical implementations of *bc* did not allow array parameters to be passed as the last parameter to a function. When *bc* was first standardized in Issue 4, this restriction was allowed. To make *bc* more widely useful, and because there are implementations without this restriction, the allowance for the restriction has been removed.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 1.3 (on page 2461), *awk*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

Updated to align with the IEEE P1003.2b draft standard, which included resolution of several interpretations of the ISO POSIX-2: 1993 standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0066 [584] and XCU/TC2-2008/0067 [679] are applied.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1230 is applied, changing the EXTENDED DESCRIPTION section to specify that array parameters can be passed as the last parameter to a function.

Austin Group Defect 1570 is applied, removing extra spacing in `"=="`.

NAME

bg — run jobs in the background

SYNOPSIS

UP

```
bg [job_id...]
```

DESCRIPTION

If job control is enabled (see the description of *set* −**m**), the shell is interactive, and the current shell execution environment (see Section 2.13, on page 2522) is not a subshell environment, the *bg* utility shall resume suspended jobs from the current shell execution environment by running them as background jobs, as described in Section 2.11 (on page 2518); it may also do so if the shell is non-interactive or the current shell execution environment is a subshell environment. If the job specified by *job_id* is already a running background job, the *bg* utility shall have no effect and shall exit successfully.

OPTIONS

None.

OPERANDS

The following operand shall be supported:

job_id Specify the job to be resumed as a background job. If no *job_id* operand is given, the most recently suspended job shall be used. The format of *job_id* is described in XBD Section 3.182 (on page 57).

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *bg*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The output of *bg* shall consist of a line in the format:

```
"[%d] %s\n", <job-number>, <command>
```

where the fields are as follows:

87756 <*job-number*> A number that can be used to identify the job to the *wait*, *fg*, and *kill* utilities. Using
87757 these utilities, the job can be identified by prefixing the job number with `'%'`.

87758 <*command*> The associated command that was given to the shell.

87759 **STDERR**

87760 The standard error shall be used only for diagnostic messages.

87761 **OUTPUT FILES**

87762 None.

87763 **EXTENDED DESCRIPTION**

87764 None.

87765 **EXIT STATUS**

87766 The following exit values shall be returned:

87767 0 Successful completion.

87768 >0 An error occurred.

87769 **CONSEQUENCES OF ERRORS**

87770 If job control is disabled, the *bg* utility shall exit with an error and no job shall be placed in the
87771 background.

87772 **APPLICATION USAGE**

87773 This utility is required to be intrinsic. See Section 1.7 (on page 2470) for details.

87774 A job is generally suspended by typing the SUSP character (<control>-Z on most systems); see
87775 XBD Chapter 11 (on page 199). At that point, *bg* can put the job into the background. This is
87776 most effective when the job is expecting no terminal input and its output has been redirected to
87777 non-terminal files. A background job can be forced to stop when it has terminal output by
87778 issuing the command:

87779 ```
stty tostop
```

87780 A background job can be stopped with the command:

87781 ```
kill -s stop job ID
```

87782 The *bg* utility does not work as expected when it is operating in its own utility execution
87783 environment because that environment has no suspended jobs. In the following examples:

87784 ```
... | xargs bg
87785 (bg)
```

87786 each *bg* operates in a different environment and does not share its parent shell's understanding
87787 of jobs. For this reason, *bg* is generally implemented as a shell regular built-in.

87788 **EXAMPLES**

87789 None.

87790 **RATIONALE**

87791 The extensions to the shell specified in this volume of POSIX.1-2024 have mostly been based on
87792 features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs* are also
87793 based on the KornShell. The standard developers examined the characteristics of the C shell
87794 versions of these utilities and found that differences exist. Despite widespread use of the C shell,
87795 the KornShell versions were selected for this volume of POSIX.1-2024 to maintain a degree of
87796 uniformity with the rest of the KornShell features selected (such as the very popular command
87797 line editing features).

  
```

The *bg* utility is expected to wrap its output if the output exceeds the number of display columns.

The *bg* and *fg* utilities are not symmetric as regards the list of process IDs known in the current shell execution environment. Whereas *fg* removes a process ID from this list, *bg* has no need to add one to this list when it resumes execution of a suspended job in the background, because this has already been done by the shell when the suspended background job was created (see Section 2.11, on page 2518).

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.9.3.1 (on page 2506), *fg*, *kill*, *jobs*, *wait*

XBD Section 3.182 (on page 57), Chapter 8 (on page 167), Chapter 11 (on page 199)

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The JC margin marker on the SYNOPSIS is removed since support for Job Control is mandatory in this version. This is a FIPS requirement.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that this utility is required to be intrinsic.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1254 is applied, updating the DESCRIPTION to account for the addition of Section 2.11 (on page 2518) and adding a paragraph to RATIONALE.

NAME

c17 — compile standard C programs

SYNOPSIS

CD

```
c17 [options...] pathname [[pathname] [-I directory]
    [-L directory] [-l library] [-R directory]]...
```

DESCRIPTION

The *c17* utility is an interface to the standard C compilation system; it shall accept source code written in the C language as defined in section 6 of the ISO C standard. The system conceptually consists of a compilation phase, encompassing Translation Phases 1 through 7 of the ISO C standard, and a linkage phase, for handling Phase 8 of the ISO C standard and extensions described here. The reference to ``library components'' in Phase 8 shall be taken to refer to components of libraries specified using the **–l** option, libraries specified as *file***.a** or *file***.so** operands, and the equivalent of a **–l c** option passed to the link editor in the manner specified in the EXTENDED DESCRIPTION. In addition, the compilation phase can be split into a separate preprocessing operation, handling Translation Phases 1 through 4, and a processing operation, handling Phases 5 though 7. Whether a single utility or multiple utilities for handling phases separately is provided by an implementation is left unspecified. The input files referenced by *pathname* operands and **–l** option-arguments shall be compiled and linked to produce an executable file or, if the **–G** option is specified, a shared library file. It is unspecified whether the linking of an executable file occurs entirely within the operation of *c17*; when a *pathname* operand or **–l** option-argument names a shared library, an executable object may be produced that is not fully resolved until the file is executed.

If the **–c** option is specified and the **–o** option is not specified, for all *pathname* operands of the form *file***.c** or *file***.i**, the files:

```
$(basename pathname .c).o
```

or

```
$(basename pathname .i).o
```

respectively shall be created as the result of successful compilation. If the **–c** option is not specified, it is unspecified whether such **.o** files are created or deleted for the *file***.c** and *file***.i** operands.

If there are no options that prevent link editing (such as **–c** or **–E**), and all input files compile and link without error, the resulting executable file or shared library file shall be written according to the **–o** *outfile* option, if present. If **–o** *outfile* is not specified, a resulting executable file shall be written to the file **a.out**; if the file to be written is a shared library file, the behavior is unspecified.

Executable files shall be created as specified in Section 1.1.1.4 (on page 2454), except that the file permission bits shall be set to:

S_IRWXO | S_IRWXG | S_IRWXU

and the bits specified by the *umask* of the process shall be cleared.

OPTIONS

The *c17* utility shall conform to XBD Section 12.2 (on page 215), except that:

- Options can be interspersed with operands.

87866 • The order of specifying the –**L**, –**l**, and –**R** options, and the order of specifying –**l** options
87867 with respect to *pathname* operands is significant.

87868 • Conforming applications shall specify each option separately; that is, grouping option
87869 letters (for example, –**cO**) need not be recognized by all implementations.

87870 The following options shall be supported:

87871 –**B** *mode* If *mode* is **dynamic**, produce a dynamically linked executable file. If the –**B** option
87872 is present with –**c**, –**E**, or –**G**, the result is unspecified.

87873 –**c** Suppress the link-edit phase of the compilation, and do not remove any object files
87874 that are produced. The application shall ensure that all operands are of the form
87875 *file***.c** or *file***.i**.

87876 –**D** *name*[**=***value*]
87877 Define *name* as if by a C-language **#define** directive. If no =*value* is given, a value of
87878 1 shall be used. The –**D** option has lower precedence than the –**U** option. That is, if
87879 *name* is used in both a –**U** and a –**D** option, *name* shall be undefined regardless of
87880 the order of the options. Additional implementation-defined *name*s may be
87881 provided by the compiler. Implementations shall support at least 2 048 bytes of –**D**
87882 definitions and 256 *names*.

87883 –**E** Copy C-language source files to standard output, executing all preprocessor
87884 directives; no compilation shall be performed. If any operand is not a text file, the
87885 effects are unspecified.

87886 –**G** Create a shared library or create object files suitable for inclusion in such a shared
87887 library. Compilations shall be performed in a manner suitable for the creation of
87888 shared libraries (for example, by producing position-independent code).

87889 If –**c** is also specified, create object files suitable for inclusion in a shared library.

87890 If –**c** is not specified, create a shared library. In this case the application shall ensure
87891 that the file named by the –**o** *outfile* option-argument includes an element named
87892 **so** or an implementation-defined element denoting a shared library, where
87893 elements in the last component of *outfile* are separated by <period> characters, for
87894 example **libx.so.1**; if no –**o** option is included in the options or the file named by
87895 the –**o** *outfile* option does not contain an element named **so** or an implementation-
87896 defined element denoting a shared library, the result is unspecified. If a *pathname*
87897 operand or –**l** option-argument names a shared library and that shared library
87898 defines an object used by the library being created, it shall become a dependency
87899 of the created shared library.

87900 If the –**G** option is present with –**B** or –**E**, the result is unspecified.

87901 –**g** Produce symbolic information in the object or executable files; the nature of this
87902 information is unspecified, and may be modified by implementation-defined
87903 interactions with other options.

87904 –**I** *directory* Change the algorithm for searching for headers whose names are not absolute
87905 pathnames to look in the directory named by the *directory* pathname before looking
87906 in the usual places. Thus, headers whose names are enclosed in double-quotes (`" "`)
87907 shall be searched for first in the directory of the file with the **#include** line, then in
87908 directories named in –**I** options, and last in the usual places. For headers whose
87909 names are enclosed in angle brackets (`"< >"`), the header shall be searched for only
87910 in directories named in –**I** options and then in the usual places. Directories named
87911 in –**I** options shall be searched in the order specified. If the –**I** option is used to

specify a directory that is one of the usual places searched by default, the results are unspecified. Implementations shall support at least ten instances of this option in a single *c17* command invocation.

–L *directory* Change the algorithm of searching for the libraries named in the **–l** objects to look in the directory named by the *directory* pathname before looking in the usual places. Directories named in **–L** options shall be searched in the order specified. If the **–L** option is used to specify a directory that is one of the usual places searched by default, the results are unspecified. Implementations shall support at least ten instances of this option in a single *c17* command invocation. If a directory specified by a **–L** option contains files with names starting with any of the strings `"libc."`, `"libl."`, `"libpthread."`, `"libm."`, `"librt."`, `"libxnet."`, or `"liby."`, the results are unspecified.

–l *library* Search the library named **lib***library***.a** or **lib***library***.so**. When searching for a library, the linker shall look at each directory specified by **–L** options that appear on the command line before this **–l** option, in the order given, and then the system default libraries. If **lib***library***.a** and **lib***library***.so** both exist in a directory, *c17* shall use **lib***library***.so** if either **–B dynamic** or **–G** is specified. Once a library has been found (shared or static) in a directory, later directories in the list shall not be considered. A library shall be searched when its name is encountered, so the placement of a **–l** option is significant. Several standard libraries can be specified in this manner, as described in the EXTENDED DESCRIPTION section. Implementations may recognize implementation-defined suffixes other than **.a** and **.so** as denoting libraries.

–O *optlevel* Specify the level of code optimization. If the *optlevel* option-argument is the digit `'0'`, all special code optimizations shall be disabled. If it is the digit `'1'`, the nature of the optimization is unspecified. If the **–O** option is omitted, the nature of the system's default optimization is unspecified. It is unspecified whether code generated in the presence of the **–O** 0 option is the same as that generated when **–O** is omitted. Other *optlevel* values may be supported.

–o *outfile* Name the output file to be produced. If the **–o** option is present with **–E**, or with **–c** and more than one input file, the result is unspecified.

When creating a single object file (by using **–c** with a single input file), use the pathname *outfile*, instead of the default *file***.o**, for the object file produced.

When creating an executable file, use the pathname *outfile*, instead of the default **a.out**, for the executable file produced.

When creating a shared library, use the pathname *outfile* as the name of the shared library. If no **–o** *outfile* option is specified when creating a shared library, the result is unspecified.

–s Produce object or executable files, or both, from which symbolic and other information not required for proper execution using the *exec* family defined in the System Interfaces volume of POSIX.1-2024 has been removed (stripped). If both **–g** and **–s** options are present, the action taken is unspecified.

–R *directory* If the object file format supports it, specify a directory to be searched for shared libraries when an executable file or shared library being created by *c17* is subsequently executed, or loaded using *dlopen*( ). If *directory* contains any <colon> or <dollar-sign> characters, the behavior is unspecified. If an implementation provides a means for setting a default load time search location or locations, the **–R**

87959 option shall take precedence.

87960 87961 The directory named by *directory* shall not be searched by a process performing dynamic loading if either of the following are true:

87962 87963 87964 87965 - The real and effective user IDs of that process are different and the directory has write permission for a user ID outside the set of the effective user ID of that process and any implementation-specific user IDs used for directories containing system libraries.

87966 87967 87968 - The real and effective group IDs of that process are different and the directory has write permission for group IDs other than the effective group ID of that process.

87969 87970 Directories named in **–R** options shall be searched in the order specified, before the default system library locations are searched.

87971 87972 87973 If a directory specified by a **–R** option contains files with names starting with any of the strings `"libc."`, `"libl."`, `"libpthread."`, `"libm."`, `"librt."`, `"libxnet."`, or `"liby."`, the result is unspecified.

87974 If the **–R** option is present with **–c** or **–E**, the result is unspecified.

87975 **–U** *name* Remove any initial definition of *name*.

87976 Multiple instances of the **–D**, **–I**, **–L**, **–l**, **–R**, and **–U** options can be specified.

87977 **OPERANDS**

87978 87979 The application shall ensure that at least one *pathname* operand is specified. The following forms for *pathname* operands shall be supported:

87980 *file*.**c** A C-language source file to be compiled and optionally linked.

87981 87982 87983 *file*.**i** A text file containing the output of *c17* **–E**, to be compiled and optionally linked. The processing already performed by *c17* **–E** when the file was produced shall not be repeated when the file is compiled.

87984 87985 87986 *file*.**a** A library of static object files typically produced by the *ar* utility, and referenced during the link-edit phase. Implementations may recognize implementation-defined suffixes other than **.a** as denoting static object file libraries.

87987 87988 87989 87990 *file*.**so** A library of shared object files typically produced by the *c17* utility with the **–G** option, and referenced during the link-edit phase. Implementations may recognize implementation-defined suffixes other than **.so** as denoting shared object file libraries.

87991 87992 87993 *file*.**o** An object file produced by *c17* **–c** and passed directly to the link editor. Implementations may recognize implementation-defined suffixes other than **.o** as denoting object files.

87994 The processing of other files is implementation-defined.

87995 **STDIN**

87996 Not used.

87997 **INPUT FILES**

87998 Each input file shall be one of the following:

87999 - A text file containing a C-language source program or the output of *c17* **–E**

- An object file in the format produced by *c17* **–c**
- A library of object files in the format produced by archiving zero or more object files using *ar*
- A shared library in the format produced by *c17* **–G**

Implementations may supply additional utilities that produce files in these formats. Additional input file formats are implementation-defined.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *c17*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

TMPDIR Provide a pathname that should override the default directory for temporary files, if any. XSI On XSI-conforming systems, provide a pathname that shall override the default directory for temporary files, if any.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If more than one *pathname* operand ending in **.c** or **.i** (or possibly other unspecified suffixes) is given, for each such file:

```
"%s:\n", <pathname>
```

may be written. These messages, if written, shall precede the processing of each input file; they shall not be written to the standard output if they are written to the standard error, as described in the STDERR section.

If the **–E** option is specified, the standard output shall be a text file that represents the results of the preprocessing stage of the language; it may contain extra information appropriate for subsequent compilation passes and shall contain at least one line with the format:

```
"# %d \"%s\"\n", <line>, <pathname>
```

for each file processed as a result of a **#include** directive, unless no other output generated from that file is present in the output, where *line* is a line number and *pathname* is the pathname used to open the file.

88039 **STDERR**

88040 The standard error shall be used only for diagnostic messages, except that if more than one
88041 *pathname* operand ending in **.c** or **.i** (or possibly other unspecified suffixes) is given, for each such
88042 file:

88043 `"%s:\n", <pathname>`

88044 may be written to allow identification of the diagnostic and warning messages with the
88045 appropriate input file. These messages, if written, shall precede the processing of each input file;
88046 they shall not be written to the standard error if they are written to the standard output, as
88047 described in the STDOUT section.

88048 This utility may produce warning messages about certain conditions that do not warrant
88049 returning an error (non-zero) exit value.

88050 **OUTPUT FILES**

88051 Object files or executable files or both are produced in unspecified formats. If the pathname of
88052 an object file or executable file to be created by *c17* resolves to an existing directory entry for a
88053 file that is not a regular file, it is unspecified whether *c17* shall attempt to create the file or shall
88054 issue a diagnostic and exit with a non-zero exit status.

88055 **EXTENDED DESCRIPTION**

88056 **Standard Libraries**

88057 The *c17* utility shall recognize the following **–l** options for standard libraries:

88058 **–l c** This option shall make available all interfaces referenced in the System Interfaces
88059 volume of POSIX.1-2024, with the possible exception of those interfaces listed as
88060 residing in **<aio.h>**, **<arpa/inet.h>**, **<complex.h>**, **<fenv.h>**, **<math.h>**,
88061 **<mqueue.h>**, **<netdb.h>**, **<net/if.h>**, **<netinet/in.h>**, **<pthread.h>**, **<sched.h>**,
88062 **<semaphore.h>**, **<spawn.h>**, **<sys/socket.h>**, **<threads.h>**, *pthread_kill*() and
88063 *pthread_sigmask*() in **<signal.h>**, interfaces marked as optional in **<sys/mman.h>**,
88064 interfaces marked as ADV (Advisory Information) in **<fcntl.h>**, and interfaces
88065 beginning with the prefix clock_ or timer_ in **<time.h>**. This option shall not be
88066 required to be present to cause a search of this library.

88067 **–l l** This option shall make available all interfaces required by the C-language output
88068 of *lex* that are not made available through the **–l c** option.

88069 **–l pthread** This option shall make available all interfaces referenced in **<pthread.h>** and
88070 **<threads.h>**, and also *pthread_kill*() and *pthread_sigmask*() referenced in
88071 **<signal.h>**. An implementation may search this library in the absence of this
88072 option.

88073 **–l m** This option shall make available all interfaces referenced in **<math.h>**,
88074 **<complex.h>**, and **<fenv.h>**. An implementation may search this library in the
88075 absence of this option.

88076 **–l rt** This option shall make available all interfaces referenced in **<aio.h>**, **<mqueue.h>**,
88077 **<sched.h>**, **<semaphore.h>**, and **<spawn.h>**, interfaces marked as optional in
88078 **<sys/mman.h>**, interfaces marked as ADV (Advisory Information) in **<fcntl.h>**,
88079 and interfaces beginning with the prefix clock_ and timer_ in **<time.h>**. An
88080 implementation may search this library in the absence of this option.

88081 **–l xnet** This option shall make available all interfaces referenced in **<arpa/inet.h>**,
88082 **<netdb.h>**, **<net/if.h>**, **<netinet/in.h>**, and **<sys/socket.h>**. An implementation
88083 may search this library in the absence of this option.

88084 **–l y** This option shall make available all interfaces required by the C-language output
88085 of *yacc* that are not made available through the **–l c** option.

88086 In the absence of options that inhibit invocation of the link editor, such as **–c** or **–E**, the *c17* utility
88087 shall cause the equivalent of a **–l c** option to be passed to the link editor after the last *pathname*
88088 operand or **–l** option, causing it to be searched after all other object files and libraries are loaded.

88089 The libraries **c**, **l**, **m**, **pthread**, **rt**, **xnet**, and **y** shall be found as shared libraries when specified as
88090 the option-argument to the **–l** option and may also be found as static libraries but, except for the
88091 shared library version of the **c** library, need not exist as regular files. The implementation may
88092 accept as **–l** option-arguments names of additional implementation-defined libraries that do not
88093 exist as regular files.

88094 **External Symbols**

88095 The C compiler and link editor shall support the significance of external symbols up to a length
88096 of at least 31 bytes; the action taken upon encountering symbols exceeding the implementation-
88097 defined maximum symbol length is unspecified.

88098 The compiler and link editor shall support a minimum of 4 095 external identifiers in one
88099 translation unit. A diagnostic message shall be written to the standard output if the
88100 implementation-defined limit is exceeded; other actions are unspecified.

88101 **Header Search**

88102 If a file with the same name as one of the standard headers defined in XBD Chapter 14 (on page
88103 221), not provided as part of the implementation, is placed in any of the usual places that are
88104 searched by default for headers, the results are unspecified.

88105 **Programming Environments**

88106 All implementations shall support one of the following programming environments as a default.
88107 Implementations may support more than one of the following programming environments.
88108 Applications can use the _POSIX_V*n*_ILP* and _POSIX_V*n*_LP* constants in **<unistd.h>**, and
88109 corresponding *sysconf*( ) and *getconf* calls, to determine which programming environments are
88110 supported.

88111 **Table 3-4** Programming Environments: Type Sizes

88112 88113 Programming Environment *getconf* Name	Bits in int	Bits in long	Bits in all pointer types	Bits in off_t
88114 _POSIX_V8_ILP32_OFF32	32	32	32	32
88115 _POSIX_V8_ILP32_OFFBIG	32	32	32	≥64
88116 _POSIX_V8_LP64_OFF64	32	64	64	64
88117 _POSIX_V8_LPBIG_OFFBIG	≥32	≥64	≥64	≥64

88118 All implementations shall support one or more environments where the widths of the following
88119 types are no greater than the width of type **long**:

88120 **blksize_t**	**ptrdiff_t**	**tcflag_t**
88121 **cc_t**	**size_t**	**wchar_t**
88122 **mode_t**	**speed_t**	**wint_t**
88123 **nfds_t**	**ssize_t**	
88124 **pid_t**	**suseconds_t**	

The executable files created when these environments are selected shall be in a proper format for execution by the *exec* family of functions. Each environment may be one of the ones in Table 3-4 (on page 2675), or it may be another environment. The names for the environments that meet this requirement shall be output by a *getconf* command using the POSIX_V8_WIDTH_RESTRICTED_ENVS argument, as a <newline>-separated list of names suitable for use with the *getconf* **–v** option. If more than one environment meets the requirement, the names of all such environments shall be output on separate lines. Any of these names can then be used in a subsequent *getconf* command to obtain the flags specific to that environment with the following suffixes added as appropriate:

_CFLAGS To get the C compiler flags.

_LDFLAGS To get the linker/loader flags.

_LIBS To get the libraries.

This requirement may be removed in a future version.

When this utility processes a file containing a function called *main*( ), it shall be defined with a return type equivalent to **int**. Using return from the initial call to *main*( ) shall be equivalent (other than with respect to language scope issues) to calling *exit*( ) with the returned value. Reaching the end of the initial call to *main*( ) shall be equivalent to calling *exit*(0). The implementation shall not declare a prototype for this function.

Implementations provide configuration strings for C compiler flags, linker/loader flags, and libraries for each supported environment. When an application needs to use a specific programming environment rather than the implementation default programming environment while compiling, the application shall first verify that the implementation supports the desired environment. If the desired programming environment is supported, the application shall then invoke *c17* with the appropriate C compiler flags as the first options for the compile, the appropriate linker/loader flags after any other options except **–l** but before any operands or **–l** options, and the appropriate libraries at the end of the operands and **–l** options.

Conforming applications shall not attempt to link together object files compiled for different programming models. Applications shall also be aware that binary data placed in shared memory or in files might not be recognized by applications built for other programming models.

Table 3-5 Programming Environments: *c17* Arguments

Programming Environment *getconf* **Name**	**Use**	*c17* **Arguments** *getconf* **Name**
_POSIX_V8_ILP32_OFF32	C Compiler Flags Linker/Loader Flags Libraries	POSIX_V8_ILP32_OFF32_CFLAGS POSIX_V8_ILP32_OFF32_LDFLAGS POSIX_V8_ILP32_OFF32_LIBS
_POSIX_V8_ILP32_OFFBIG	C Compiler Flags Linker/Loader Flags Libraries	POSIX_V8_ILP32_OFFBIG_CFLAGS POSIX_V8_ILP32_OFFBIG_LDFLAGS POSIX_V8_ILP32_OFFBIG_LIBS
_POSIX_V8_LP64_OFF64	C Compiler Flags Linker/Loader Flags Libraries	POSIX_V8_LP64_OFF64_CFLAGS POSIX_V8_LP64_OFF64_LDFLAGS POSIX_V8_LP64_OFF64_LIBS
_POSIX_V8_LPBIG_OFFBIG	C Compiler Flags Linker/Loader Flags Libraries	POSIX_V8_LPBIG_OFFBIG_CFLAGS POSIX_V8_LPBIG_OFFBIG_LDFLAGS POSIX_V8_LPBIG_OFFBIG_LIBS

In addition to the type size programming environments above, all implementations also support a multi-threaded programming environment that is orthogonal to all of the programming environments listed above. The *getconf* utility can be used to get flags for the threaded programming environment, as indicated in Table 3-6.

Table 3-6 Threaded Programming Environment: *c17* Arguments

Programming Environment *getconf* **Name**	**Use**	*c17* **Arguments** *getconf* **Name**
_POSIX_THREADS	C Compiler Flags	POSIX_V8_THREADS_CFLAGS
	Linker/Loader Flags	POSIX_V8_THREADS_LDFLAGS

These programming environment flags may be used in conjunction with any of the type size programming environments supported by the implementation.

EXIT STATUS

The following exit values shall be returned:

0 Successful compilation or link edit.

>0 An error occurred.

CONSEQUENCES OF ERRORS

When *c17* encounters a compilation error that causes an object file not to be created, it shall write a diagnostic to standard error and continue to compile other source code operands, but it shall not perform the link phase and it shall return a non-zero exit status. If the link edit is unsuccessful, a diagnostic message shall be written to standard error and *c17* exits with a non-zero status. A conforming application shall rely on the exit status of *c17*, rather than on the existence or mode of the executable file.

APPLICATION USAGE

Since the *c17* utility usually creates files in the current directory during the compilation process, it is typically necessary to run the *c17* utility in a directory in which a file can be created.

On systems providing POSIX Conformance (see XBD Chapter 2, on page 15), *c17* is required only with the C-Language Development option; XSI-conformant systems always provide *c17*.

Since this standard requires that conforming applications define either _POSIX_C_SOURCE or _XOPEN_SOURCE before inclusion of any header (see XSH Section 2.2.1, on page 496), if *c17* is used to compile source code that includes a header without defining one of these feature test macros in the required manner, the behavior of *c17* itself and the results of using any files it generates are undefined. When *c17* is used this way, implementations are encouraged to make visible in headers from the ISO C standard only the symbols that are allowed by that standard, and otherwise to behave the same as if _POSIX_C_SOURCE or _XOPEN_SOURCE had been defined, but portable applications cannot rely on this.

Some historical implementations have created **.o** files when **−c** is not specified and more than one source file is given. Since this area is left unspecified, the application cannot rely on **.o** files being created, but it also must be prepared for any related **.o** files that already exist being deleted at the completion of the link edit.

There is the possible implication that if a user supplies versions of the standard functions (before they would be encountered by an implicit **−l c** or explicit **−l m**), that those versions would be used in place of the standard versions. There are various reasons this might not be true (functions defined as macros, manipulations for clean name space, and so on), so the existence of files named in the same manner as the standard libraries within the **−L** directories is explicitly

stated to produce unspecified behavior.

All of the functions specified in the System Interfaces volume of POSIX.1-2024 may be made visible by implementations when the Standard C Library is searched. Conforming applications must explicitly request searching the other standard libraries when functions made visible by those libraries are used.

In the ISO C standard the mapping from physical source characters to the C source character set is implementation-defined. Implementations may strip white-space characters before the terminating <newline> of a (physical) line as part of this mapping and, as a consequence of this, one or more white-space characters (and no other characters) between a <backslash> character and the <newline> character that terminates the line produces implementation-defined results. Portable applications should not use such constructs.

Some *c17* compilers not conforming to POSIX.1-2024 do not support trigraphs by default.

Implementations may support multiple programming environments with some of them conforming to this standard and some not conforming. The _POSIX_V*n*_ILP* and _POSIX_V*n*_LP* constants in **<unistd.h>**, and corresponding *sysconf*( ) and *getconf* calls, only indicate whether each programming environment is supported; they do not indicate anything about conformance of the environments that are supported. For example, an implementation may support the ILP32_OFF32 environment for legacy reasons with a 32-bit **time_t**, whereas in a conforming environment **time_t** is required to have a width of at least 64 bits. Application writers should consult an implementation's POSIX Conformance Document for information about the conformance of each supported programming environment.

EXAMPLES

1. The following usage example compiles **foo.c** and creates the executable file **foo**:

   ```
   c17 -o foo foo.c
   ```

   The following usage example compiles **foo.c** and creates the object file **foo.o**:

   ```
   c17 -c foo.c
   ```

   The following usage example compiles **foo.c** and creates the executable file **a.out**:

   ```
   c17 foo.c
   ```

   The following usage example compiles **foo.c**, links it with **bar.o**, and creates the executable file **a.out**. It may also create and leave **foo.o**:

   ```
   c17 foo.c bar.o
   ```

   The following usage example preprocesses **foo.c** and **bar.c** to create **foo.i** and **bar.i**, compiles **foo.i** and **bar.i** to create **foo.o** and **bar.o**, then links **foo.o** and **bar.o** to create the executable file **foobar**. Each *c17* execution would ideally be invoked from a separate rule in a makefile (see *make*, on page 3130) with suitable dependencies so that each is only executed when it needs to be:

   ```
   c17 -E foo.c > foo.i
   c17 -E bar.c > bar.i
   c17 -c foo.i
   c17 -c bar.i
   c17 -o foobar foo.o bar.o
   ```

2. The following example shows how an application using threads interfaces can test for support of and use a programming environment supporting 32-bit **int**, **long**, and all pointer types, and an **off_t** type using at least 64 bits:


```
offbig_env=$(getconf _POSIX_V8_ILP32_OFFBIG)
if [ $offbig_env != "-1" ] && [ $offbig_env != "undefined" ]
then
    c17 $(getconf POSIX_V8_ILP32_OFFBIG_CFLAGS) \
    $(getconf POSIX_V8_THREADS_CFLAGS) -D_XOPEN_SOURCE=800 \
    $(getconf POSIX_V8_ILP32_OFFBIG_LDFLAGS) \
    $(getconf POSIX_V8_THREADS_LDFLAGS) foo.c -o foo \
    $(getconf POSIX_V8_ILP32_OFFBIG_LIBS) \
    -l pthread
else
    echo ILP32_OFFBIG programming environment not supported
    exit 1
fi
```

3. The following examples clarify the use and interactions of **–L** and **–l** options.

   Consider the case in which module **a.c** calls function *f*( ) in library **libQ.a**, and module **b.c** calls function *g*( ) in library **libp.a**. Assume that both libraries reside in **/a/b/c**. The command line to compile and link in the desired way is:

   ```
   c17 -L /a/b/c main.o a.c -l Q b.c -l p
   ```

   In this case the **–L** option need only precede the first **–l** option, since both **libQ.a** and **libp.a** reside in the same directory.

   Multiple **–L** options can be used when library name collisions occur. Building on the previous example, suppose that the user wants to use a new **libp.a**, in **/a/a/a**, but still wants *f*( ) from **/a/b/c/libQ.a**:

   ```
   c17 -L /a/a/a -L /a/b/c main.o a.c -l Q b.c -l p
   ```

   In this example, the linker searches the **–L** options in the order specified, and finds **/a/a/a/libp.a** before **/a/b/c/libp.a** when resolving references for **b.c**. The order of the **–l** options is still important, however.

4. The following example shows how an application can use a programming environment where the widths of the following types:

   blksize_t, **cc_t**, **mode_t**, **nfds_t**, **pid_t**, **ptrdiff_t**, **size_t**, **speed_t**, **ssize_t**, **suseconds_t**, **tcflag_t**, **wchar_t**, **wint_t**

   are no greater than the width of type **long**:

   ```
   # First choose one of the listed environments ...
   # ... if there are no additional constraints, the first one will do:
   CENV=$(getconf POSIX_V8_WIDTH_RESTRICTED_ENVS | head -n 1)
   # ... or, if an environment that supports large files is preferred,
   # look for names that contain "OFF64" or "OFFBIG". (This chooses
   # the last one in the list if none match.)
   for CENV in $(getconf POSIX_V8_WIDTH_RESTRICTED_ENVS)
   do
       case $CENV in
       *OFF64*|*OFFBIG*) break ;;
       esac
   done
   ```



```
# The chosen environment name can now be used like this:

c17 $(getconf ${CENV}_CFLAGS) -D _POSIX_C_SOURCE=202405L \
$(getconf ${CENV}_LDFLAGS) foo.c -o foo \
$(getconf ${CENV}_LIBS)
```

5. The following example shows how to create a shared library that does not depend on any other shared library:

```
c17 -G -c foo.c bar.c
c17 -G -o foobar.so foo.o bar.o
```

6. The following example shows how to create a dynamic executable that loads application specific shared libraries by searching a specified list of directories when it is executed:

```
c17 -G -c foo.c
c17 -G -o /path/to/dir1/foo.so foo.o
c17 -G -c bar.c
c17 -G -o /path/to/dir2/bar.so bar.o
c17 -B dynamic -L /path/to/dir1 -L /path/to/dir2 -R /path/to/dir1 \
    -R /path/to/dir2 -o foobar foobar.c -l foo -l bar
```

RATIONALE

The *c17* utility is based on the *c89* utility originally introduced in the ISO POSIX-2: 1993 standard.

Some of the changes from *c89* include the ability to intersperse options and operands (which many *c89* implementations allowed despite it not being specified), the description of **–l** as an option instead of an operand, and the modification to the contents of the Standard Libraries section to account for new headers and options; for example, **<spawn.h>** added to the description of **–l rt**.

POSIX.1-2024 specifies that the *c17* utility must be able to use regular files for ***.o** files and for **a.out** files. Implementations are free to overwrite existing files of other types when attempting to create object files and executable files, but are not required to do so. If something other than a regular file is specified and using it fails for any reason, *c17* is required to issue a diagnostic message and exit with a non-zero exit status. But for some file types, the problem may not be noticed for a long time. For example, if a FIFO named **a.out** exists in the current directory, *c17* may attempt to open **a.out** and will hang in the *open*( ) call until another process opens the FIFO for reading. Then *c17* may write most of the **a.out** to the FIFO and fail when it tries to seek back close to the start of the file to insert a timestamp (FIFOs are not seekable files). The *c17* utility is also allowed to issue a diagnostic immediately if it encounters an **a.out** or ***.o** file that is not a regular file. For portable use, applications should ensure that any **a.out**, **–o** option-argument, or ***.o** files corresponding to any ***.c** files do not conflict with names already in use that are not regular files or symbolic links that point to regular files.

Commands of the form `c17 -c -o ...` are frequently used to directly place the **.o** file into an alternative directory without a need to separately rename the output file. This helps to support concurrent compilations and out of tree builds.

Some implementations allow **–c** **–o** *directory* to produce *directory*/*file*.**o** even when there is more than one input file; however, portable applications using **–c** with **–o** must compile only one file at a time and must specify the final destination filename rather than a directory.

The shared library version of the **c** library is required to exist as a regular file because the dynamic linker needs to be able to load at least one library at execution time. Other standard shared libraries need not exist in their own right if the interfaces the standard requires them to

provide exist in the **c** library; all that is required is that they are ``found'' when specified as **–l** option-arguments. Static versions of the standard libraries need not exist as regular files, even if they are found as static libraries when specified as **–l** option-arguments.

On many systems, multi-threaded applications run in a programming environment that is distinct from that used by single-threaded applications. This multi-threaded programming environment (in addition to needing to specify **–l pthread** at link time) may require additional flags to be set when headers are processed at compile time (**–D_REENTRANT** being common). This programming environment is orthogonal to the type size programming environments discussed above and listed in Table 3-4 (on page 2675). This version of the standard adds *getconf* utility calls to provide the C compiler flags and linker/loader flags needed to support multi-threaded applications. Note that on a system where single-threaded applications are a special case of a multi-threaded application, both of these *getconf* calls may return NULL strings; on other implementations both of these strings may be non-NULL strings.

The C standardization committee invented trigraphs (e.g., `"??!"` to represent `'|'`) to address character portability problems in development environments based on national variants of the 7-bit ISO/IEC 646:1991 standard character set. However, these environments were already obsolete by the time the first ISO C standard was published, and in practice trigraphs have not been used for their intended purpose, and usually are intended to have their original meaning in K&R C. For example, in practice a C-language source string like `"What??!"` is usually intended to end in two <question-mark> characters and an <exclamation-mark>, not in `'|'`.

When the **–E** option is used, execution of some **#pragma** preprocessor directives may simply result in a copy of the directive being included in the output as part of the allowed extra information used by subsequent compilation passes (see STDOUT).

This standard requires that, when **–E** is used, lines beginning with `'#'` are output identifying the files processed as a result of **#include** directives in order that *c17* **–E** can be used to generate makefile dependencies. See *make*. Usually such lines are output each time the origin of the subsequent lines in the output changes, and the line number after the `'#'` is the line number in the named file of the line from which the next line in the output was derived.

When the **–R** option is not included when an executable file or shared library is being created, some implementations use the environment variables *LD_RUN_PATH* and *LD_LIBRARY_PATH* to determine the directories to be searched for shared libraries.

Some implementations permit place-holders preceded by a <dollar-sign> character (`'$'`), such as $ORIGIN, in the **–R** directory option-argument to be evaluated at load time. Some implementations accept a colon separated list of directories for the path to search for shared libraries, with the same effect as specifying the **–R** option multiple times. However, these features are not universal.

The name of a shared library usually contains an element named **so**. Other implementation-defined elements are allowed for backwards compatibility with historical systems, and so that tools can be developed on conforming systems to create libraries for multiple environments. For example, Microsoft systems use the filename extension **.dll** (and do not allow following text) to denote a shared library. The standard allows additional characters to be used in the name of a library following an **so** element to permit shared library versioning information to be at the end of the library filename rather than requiring that any such strings appear before the final element of the library name.

The decision to standardize on **so** as a required element in a shared library name was intentional, as the alternative would have been standardizing things such as a new make macro $(SHLIB_EXT) that would otherwise be needed to write a portable makefile that can compile shared libraries despite not having a standardized element name.

If a combination of direct and indirect dependencies of a shared library would require different versions of another shared library, options that are not specified by the standard (such as **–B direct**) will probably need to be used when linking that shared library, so that at runtime the intended versions are found.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

Unlike all of the other non-OB-shaded utilities in this standard, a utility by this name probably will not appear in the next version of this standard. This utility's name is tied to the current revision of the ISO C standard at the time this standard is approved. Since the ISO C standard and this standard are maintained by different organizations on different schedules, we cannot predict what the compiler will be named in the next version of the standard.

SEE ALSO

Section 1.1.1.4 (on page 2454), *ar*, *getconf*, *make*, *nm*, *strip*, *umask*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215), Chapter 14 (on page 221)

XSH *exec*, *sysconf*( )

CHANGE HISTORY

First released in Issue 8. Included for alignment with the ISO/IEC 9899: 2018 standard.

NAME

cal — print a calendar

SYNOPSIS

XSI
```
cal [[month] year]
```

DESCRIPTION

The *cal* utility shall write a calendar to standard output using the Julian calendar for dates from January 1, 1 through September 2, 1752 and the Gregorian calendar for dates from September 14, 1752 through December 31, 9999 as though the Gregorian calendar had been adopted on September 14, 1752.

If no operands are given, *cal* shall produce a one-month calendar for the current month in the current year. If only the *year* operand is given, *cal* shall produce a calendar for all twelve months in the given calendar year. If both *month* and *year* operands are given, *cal* shall produce a one-month calendar for the given month in the given year.

OPTIONS

None.

OPERANDS

The following operands shall be supported:

month Specify the month to be displayed, represented as a decimal integer from 1 (January) to 12 (December).

year Specify the year for which the calendar is displayed, represented as a decimal integer from 1 to 9999.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *cal*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error, and informative messages written to standard output.

LC_TIME Determine the format and contents of the calendar.

NLSPATH Determine the location of messages objects and message catalogs.

TZ Determine the timezone used to calculate the value of the current month.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall be used to display the calendar, in an unspecified format.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Note that:

```
cal 83
```

refers to A.D. 83, not 1983.

EXAMPLES

None.

RATIONALE

Earlier versions of this standard incorrectly required that the command:

```
cal 2000
```

write a one-month calendar for the current calendar month (no matter what the current year is) in the year 2000 to standard output. This did not match historic practice in any known version of the *cal* utility. The description has been updated to match historic practice. When only the *year* operand is given, *cal* writes a twelve-month calendar for the specified year.

FUTURE DIRECTIONS

A future version of this standard may support locale-specific recognition of the date of adoption of the Gregorian calendar.

SEE ALSO

XBD Chapter 8 (on page 167)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The DESCRIPTION is updated to allow for traditional behavior for years before the adoption of the Gregorian calendar.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0074 [56] and XCU/TC1-2008/0075 [56] are applied.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

88505 **NAME**

88506 cat — concatenate and print files

88507 **SYNOPSIS**

88508 `cat [-u] [file...]`

88509 **DESCRIPTION**

88510 The *cat* utility shall read files in sequence and shall write their contents to the standard output in
88511 the same sequence.

88512 **OPTIONS**

88513 The *cat* utility shall conform to XBD Section 12.2 (on page 215).

88514 The following option shall be supported:

88515 **−u** Write bytes from the input file to the standard output without delay as each is
88516 read.

88517 **OPERANDS**

88518 The following operand shall be supported:

88519 *file* A pathname of an input file. If no *file* operands are specified, the standard input
88520 shall be used. If a *file* is `'-'`, the *cat* utility shall read from the standard input at
88521 that point in the sequence. The *cat* utility shall not close and reopen standard input
88522 when it is referenced in this way, but shall accept multiple occurrences of `'-'` as a
88523 *file* operand.

88524 **STDIN**

88525 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is `'-'`.
88526 See the INPUT FILES section.

88527 **INPUT FILES**

88528 The input files can be any file type.

88529 **ENVIRONMENT VARIABLES**

88530 The following environment variables shall affect the execution of *cat*:

88531 *LANG* Provide a default value for the internationalization variables that are unset or null.
88532 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
88533 variables used to determine the values of locale categories.)

88534 *LC_ALL* If set to a non-empty string value, override the values of all the other
88535 internationalization variables.

88536 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
88537 characters (for example, single-byte as opposed to multi-byte characters in
88538 arguments).

88539 *LC_MESSAGES*
88540 Determine the locale that should be used to affect the format and contents of
88541 diagnostic messages written to standard error.

88542 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

88543 **ASYNCHRONOUS EVENTS**

88544 Default.

88545 **STDOUT**

88546 The standard output shall contain the sequence of bytes read from the input files. Nothing else
88547 shall be written to the standard output. If the standard output is a regular file, and is the same
88548 file as any of the input file operands, the implementation may treat this as an error.

88549 **STDERR**

88550 The standard error shall be used only for diagnostic messages.

88551 **OUTPUT FILES**

88552 None.

88553 **EXTENDED DESCRIPTION**

88554 None.

88555 **EXIT STATUS**

88556 The following exit values shall be returned:

88557 0 All input files were output successfully.

88558 >0 An error occurred.

88559 **CONSEQUENCES OF ERRORS**

88560 Default.

88561 **APPLICATION USAGE**

88562 The **–u** option has value in prototyping non-blocking reads from FIFOs. The intent is to support
88563 the following sequence:

```
88564 mkfifo foo
88565 cat –u foo > /dev/tty13 &
88566 cat –u > foo
```

88567 It is unspecified whether standard output is or is not buffered in the default case. This is
88568 sometimes of interest when standard output is associated with a terminal, since buffering may
88569 delay the output. The presence of the **–u** option guarantees that unbuffered I/O is available. It is
88570 implementation-defined whether the *cat* utility buffers output if the **–u** option is not specified.
88571 Traditionally, the **–u** option is implemented using the equivalent of the *setvbuf*( ) function
88572 defined in the System Interfaces volume of POSIX.1-2024.

88573 **EXAMPLES**

88574 The following command:

```
88575 cat myfile
```

88576 writes the contents of the file **myfile** to standard output.

88577 The following command:

```
88578 cat doc1 doc2 > doc.all
```

88579 concatenates the files **doc1** and **doc2** and writes the result to **doc.all**.

88580 Because of the shell language mechanism used to perform output redirection, a command such
88581 as this:

```
88582 cat doc doc.end > doc
```

88583 causes the original data in **doc** to be lost before *cat* even begins execution. This is true whether
88584 the *cat* command fails with an error or silently succeeds (the specification allows both
88585 behaviors). In order to append the contents of **doc.end** without losing the original contents of
88586 **doc**, this command should be used instead:


```
cat doc.end >> doc
```

The command:

```
cat start - middle - end > file
```

when standard input is a terminal, gets two arbitrary pieces of input from the terminal with a single invocation of *cat*. Note, however, that if standard input is a regular file, this would be equivalent to the command:

```
cat start - middle /dev/null end > file
```

because the entire contents of the file would be consumed by *cat* the first time `'-'` was used as a *file* operand and an end-of-file condition would be detected immediately when `'-'` was referenced the second time.

RATIONALE

Historical versions of the *cat* utility include the **–e**, **–t**, and **–v**, options which permit the ends of lines, **<**tab**>** characters, and invisible characters, respectively, to be rendered visible in the output. The standard developers omitted these options because they provide too fine a degree of control over what is made visible, and similar output can be obtained using a command such as:

```
sed -n l pathname
```

The latter also has the advantage that its output is unambiguous, whereas the output of historical *cat* **–etv** is not.

The **–s** option was omitted because it corresponds to different functions in BSD and System V-based systems. The BSD **–s** option to squeeze blank lines can be accomplished by the shell script shown in the following example:

```
sed -n '
# Write non-empty lines.
/./   {
      p
      d
      }
# Write a single empty line, then look for more empty lines.
/^$/  p
# Get next line, discard the held <newline> (empty line),
# and look for more empty lines.
:Empty
/^$/  {
      N
      s/.//
      b Empty
      }
# Write the non-empty line before going back to search
# for the first in a set of empty lines.
      p
'
```

The System V **–s** option to silence error messages can be accomplished by redirecting the standard error. Note that the BSD documentation for *cat* uses the term ``blank line'' to mean the same as the POSIX ``empty line'': a line consisting only of a **<**newline**>**.

The BSD **–n** option was omitted because similar functionality can be obtained from the **–n** option of the *pr* utility.

88633 **FUTURE DIRECTIONS**

88634 None.

88635 **SEE ALSO**

88636 *more*

88637 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

88638 XSH *setvbuf*( )

88639 **CHANGE HISTORY**

88640 First released in Issue 2.

88641 **Issue 7**

88642 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

88643 SD5-XCU-ERN-174 is applied, changing the RATIONALE.

88644 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0073 [876] is applied.

88645 **Issue 8**

88646 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

88647 **NAME**

88648 cd — change the working directory

88649 **SYNOPSIS**

88650 `cd [-L] [`*directory*`]`

88651 `cd -P [-e] [`*directory*`]`

88652 **DESCRIPTION**

88653 The *cd* utility shall change the working directory of the current shell execution environment (see
88654 Section 2.13, on page 2522) by executing the following steps in sequence. (In the following steps,
88655 the symbol **curpath** represents an intermediate value used to simplify the description of the
88656 algorithm used by *cd*. There is no requirement that **curpath** be made visible to the application.)

88657 1. If no *directory* operand is given and the *HOME* environment variable is empty or
88658 undefined, the default behavior is implementation-defined and no further steps shall be
88659 taken.

88660 2. If no *directory* operand is given and the *HOME* environment variable is set to a non-empty
88661 value, the *cd* utility shall behave as if the directory named in the *HOME* environment
88662 variable was specified as the *directory* operand.

88663 3. If the *directory* operand begins with a <slash> character, set **curpath** to the operand and
88664 proceed to step 7.

88665 4. If the first component of the *directory* operand is dot or dot-dot, proceed to step 6.

88666 5. Starting with the first pathname in the <colon>-separated pathnames of *CDPATH* (see the
88667 ENVIRONMENT VARIABLES section) if the pathname is non-null, test if the
88668 concatenation of that pathname, a <slash> character if that pathname did not end with a
88669 <slash> character, and the *directory* operand names a directory. If the pathname is null,
88670 test if the concatenation of dot, a <slash> character, and the operand names a directory. In
88671 either case, if the resulting string names an existing directory, set **curpath** to that string
88672 and proceed to step 7. Otherwise, repeat this step with the next pathname in *CDPATH*
88673 until all pathnames have been tested.

88674 6. Set **curpath** to the *directory* operand.

88675 7. If the **–P** option is in effect, proceed to step 10. If **curpath** does not begin with a <slash>
88676 character, set **curpath** to the string formed by the concatenation of the value of *PWD*, a
88677 <slash> character if the value of *PWD* did not end with a <slash> character, and **curpath**.

88678 8. The **curpath** value shall then be converted to canonical form as follows, considering each
88679 component from beginning to end, in sequence:

88680 a. Dot components and any <slash> characters that separate them from the next
88681 component shall be deleted.

88682 b. For each dot-dot component, if there is a preceding component and it is neither
88683 root nor dot-dot, then:

88684 i. If the preceding component does not refer (in the context of pathname
88685 resolution with symbolic links followed) to a directory, then the *cd* utility
88686 shall display an appropriate error message and no further steps shall be
88687 taken.

88688 ii. The preceding component, all <slash> characters separating the preceding
88689 component from dot-dot, dot-dot, and all <slash> characters separating dot-
88690 dot from the following component (if any) shall be deleted.

c. An implementation may further simplify **curpath** by removing any trailing <slash> characters that are not also leading <slash> characters, replacing multiple non-leading consecutive <slash> characters with a single <slash>, and replacing three or more leading <slash> characters with a single <slash>. If, as a result of this canonicalization, the **curpath** variable is null, no further steps shall be taken.

9. If **curpath** is longer than {PATH_MAX} bytes (including the terminating null) and the *directory* operand was not longer than {PATH_MAX} bytes (including the terminating null), then **curpath** shall be converted from an absolute pathname to an equivalent relative pathname if possible. This conversion shall always be considered possible if the value of *PWD*, with a trailing <slash> added if it does not already have one, is an initial substring of **curpath**. Whether or not it is considered possible under other circumstances is unspecified. Implementations may also apply this conversion if **curpath** is not longer than {PATH_MAX} bytes or the *directory* operand was longer than {PATH_MAX} bytes.

10. The *cd* utility shall then perform actions equivalent to the *chdir*( ) function called with **curpath** as the *path* argument. If these actions fail for any reason, the *cd* utility shall display an appropriate error message and the remainder of this step shall not be executed. If the **−P** option is not in effect, the *PWD* environment variable shall be set to the value that **curpath** had on entry to step 9 (i.e., before conversion to a relative pathname).

    If the **−P** option is in effect, the *PWD* environment variable shall be set to the string that would be output by *pwd* **−P**. If there is insufficient permission on the new directory, or on any parent of that directory, to determine the current working directory, the value of the *PWD* environment variable is unspecified. If both the **−e** and the **−P** options are in effect and *cd* is unable to determine the pathname of the current working directory, *cd* shall complete successfully but return a non-zero exit status.

If, during the execution of the above steps, the *PWD* environment variable is set, the *OLDPWD* shell variable shall also be set to the value of the old working directory (that is the current working directory immediately prior to the call to *cd*). It is unspecified whether, when setting *OLDPWD*, the shell also causes it to be exported if it was not already.

OPTIONS

The *cd* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported by the implementation:

−e If the **−P** option is in effect, the current working directory is successfully changed, and the correct value of the *PWD* environment variable cannot be determined, exit with exit status 1.

−L Handle the operand dot-dot logically; symbolic link components shall not be resolved before dot-dot components are processed (see steps 8. and 9. in the DESCRIPTION).

−P Handle the operand dot-dot physically; symbolic link components shall be resolved before dot-dot components are processed (see step 7. in the DESCRIPTION).

If both **−L** and **−P** options are specified, the last of these options shall be used and all others ignored. If neither **−L** nor **−P** is specified, the operand shall be handled dot-dot logically; see the DESCRIPTION.

88735 **OPERANDS**

88736 The following operands shall be supported:

88737 *directory* An absolute or relative pathname of the directory that shall become the new
88738 working directory. The interpretation of a relative pathname by *cd* depends on the
88739 **–L** option and the *CDPATH* and *PWD* environment variables. If *directory* is an
88740 empty string, *cd* shall write a diagnostic message to standard error and exit with
88741 non-zero status. If *directory* consists of a single `'-'` (<hyphen-minus>) character,
88742 the *cd* utility shall behave as if *directory* contained the value of the *OLDPWD*
88743 environment variable, except that after it sets the value of *PWD* it shall write the
88744 new value to standard output. The behavior is unspecified if *OLDPWD* does not
88745 start with a <slash> character.

88746 **STDIN**

88747 Not used.

88748 **INPUT FILES**

88749 None.

88750 **ENVIRONMENT VARIABLES**

88751 The following environment variables shall affect the execution of *cd*:

88752 *CDPATH* A <colon>-separated list of pathnames that refer to directories. The *cd* utility shall
88753 use this list in its attempt to change the directory, as described in the
88754 DESCRIPTION. An empty string in place of a directory pathname represents the
88755 current directory. If *CDPATH* is not set, it shall be treated as if it were an empty
88756 string.

88757 *HOME* The name of the directory, used when no *directory* operand is specified.

88758 *LANG* Provide a default value for the internationalization variables that are unset or null.
88759 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
88760 variables used to determine the values of locale categories.)

88761 *LC_ALL* If set to a non-empty string value, override the values of all the other
88762 internationalization variables.

88763 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
88764 characters (for example, single-byte as opposed to multi-byte characters in
88765 arguments).

88766 *LC_MESSAGES*

88767 Determine the locale that should be used to affect the format and contents of
88768 diagnostic messages written to standard error.

88769 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

88770 *OLDPWD* A pathname of the previous working directory, used when the operand is `'-'`. If
88771 an application sets or unsets the value of *OLDPWD*, the behavior of *cd* with a `'-'`
88772 operand is unspecified.

88773 *PWD* This variable shall be set as specified in the DESCRIPTION. If an application sets
88774 or unsets the value of *PWD*, the behavior of *cd* is unspecified.

88775 **ASYNCHRONOUS EVENTS**

88776 Default.

88777 **STDOUT**

88778 If a non-empty directory name from *CDPATH* is used, or if the operand `'-'` is used, and the
88779 absolute pathname of the new working directory can be determined, that pathname shall be
88780 written to the standard output as follows:

88781 `"%s\n", <new directory>`

88782 If an absolute pathname of the new current working directory cannot be determined, it is
88783 unspecified whether nothing is written to the standard output or the value of **curpath** used in
88784 step 10, followed by a <newline>, is written to the standard output.

88785 If a non-empty directory name from *CDPATH* is not used, and the directory argument is not
88786 `'-'`, there shall be no output.

88787 **STDERR**

88788 The standard error shall be used only for diagnostic messages.

88789 **OUTPUT FILES**

88790 None.

88791 **EXTENDED DESCRIPTION**

88792 None.

88793 **EXIT STATUS**

88794 The following exit values shall be returned:

88795 0 The current working directory was successfully changed and the value of the *PWD*
88796 environment variable was set correctly.

88797 0 The current working directory was successfully changed, the **−e** option is not in effect, the
88798 **−P** option is in effect, and the correct value of the *PWD* environment variable could not be
88799 determined.

88800 >0 Either the **−e** option or the **−P** option is not in effect, and an error occurred.

88801 1 The current working directory was successfully changed, both the **−e** and the **−P** options are
88802 in effect, and the correct value of the *PWD* environment variable could not be determined.

88803 >1 Both the **−e** and the **−P** options are in effect, and an error occurred.

88804 **CONSEQUENCES OF ERRORS**

88805 The working directory shall remain unchanged.

88806 **APPLICATION USAGE**

88807 This utility is required to be intrinsic. See Section 1.7 (on page 2470) for details.

88808 Since *cd* affects the current shell execution environment, it is always provided as a shell regular
88809 built-in. If it is called in a subshell or separate utility execution environment, such as one of the
88810 following:

```
88811 (cd /tmp)
88812 nohup cd
88813 find . -exec cd {} \;
```

88814 it does not affect the working directory of the caller's environment.

88815 The user must have execute (search) permission in *directory* in order to change to it.

88816 Since *cd* treats the operand `'-'` as a special case, applications should not pass arbitrary values as
88817 the operand. For example, instead of:

```
88818 CDPATH= cd -P -- "$dir"
```


applications should use the following:

```
case $dir in
(/*) cd -P "$dir";;
("") echo >&2 directory is an empty string; exit 1;;
(*) CDPATH= cd -P "./$dir";;
esac
```

If an absolute pathname of the new current working directory cannot be determined, and a non-empty directory name from *CDPATH* is used, *cd* may write a pathname to standard output that is not an absolute pathname.

EXAMPLES

The following template can be used to perform processing in the directory specified by *location* and end up in the current working directory in use before the first *cd* command was issued:

```
cd location
if [ $? -ne 0 ]
then
    print error message
    exit 1
fi
... do whatever is desired as long as the OLDPWD environment variable
    is not modified
cd -
```

RATIONALE

The use of the *CDPATH* was introduced in the System V shell. Its use is analogous to the use of the *PATH* variable in the shell. The BSD C shell used a shell parameter *cdpath* for this purpose.

A common extension when *HOME* is undefined is to get the login directory from the user database for the invoking user. This does not occur on System V implementations.

Some historical shells, such as the KornShell, took special actions when the directory name contained a dot-dot component, selecting the logical parent of the directory, rather than the actual parent directory; that is, it moved up one level toward the '/' in the pathname, remembering what the user typed, rather than performing the equivalent of:

```
chdir("..");
```

In such a shell, the following commands would not necessarily produce equivalent output for all directories:

```
cd .. && ls      ls ..
```

This behavior is now the default. It is not consistent with the definition of dot-dot in most historical practice; that is, while this behavior has been optionally available in the KornShell, other shells have historically not supported this functionality. The logical pathname is stored in the *PWD* environment variable when the *cd* utility completes and this value is used to construct the next directory name if *cd* is invoked with the **-L** option.

When the **-P** option is in effect, the correct value of the *PWD* environment variable cannot be determined on some systems, but still results in a zero exit status. The value of *PWD* doesn't matter to some shell scripts and in those cases this is not a problem. In other cases, especially with multiple calls to *cd*, the values of *PWD* and *OLDPWD* are important but the standard provided no easy way to know that this was the case. The **-e** option has been added, even though this was not historic practice, to give script writers a reliable way to know when the value of *PWD* is not reliable.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

Section 2.13 (on page 2522), *pwd*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *chdir*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The *cd* – operand, *PWD*, and *OLDPWD* are added.

The **–L** and **–P** options are added to align with the IEEE P1003.2b draft standard. This also includes the introduction of a new description to include the effect of these options.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/14 is applied, changing the SYNOPSIS to make it clear that the **–L** and **–P** options are mutually-exclusive.

Issue 7

Austin Group Interpretation 1003.1-2001 #037 is applied.

Austin Group Interpretation 1003.1-2001 #199 is applied, clarifying how the *cd* utility handles concatenation of two pathnames when the first pathname ends in a <slash> character.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Step 7 of the processing performed by *cd* is revised to refer to **curpath** instead of ``the operand''.

Changes to the *pwd* utility and *PWD* environment variable have been made to match the changes to the *getcwd*( ) function made for Austin Group Interpretation 1003.1-2001 #140.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0076 [230], XCU/TC1-2008/0077 [240], XCU/TC1-2008/0078 [240], and XCU/TC1-2008/0079 [123] are applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0074 [584] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 253 is applied, adding the **–e** option.

Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that this utility is required to be intrinsic.

Austin Group Defect 1045 is applied, clarifying the behavior when the *directory* operand is `'-'`.

Austin Group Defect 1047 is applied, requiring *cd* to treat an empty *directory* operand as an error

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1527 is applied, clarifying the behavior when an absolute pathname of the new current working directory cannot be determined.

Austin Group Defect 1601 is applied, clarifying that when setting *OLDPWD*, the shell need not cause it to be exported if it was not already.

NAME

cflow — generate a C-language flowgraph (**DEVELOPMENT**)

SYNOPSIS

XSI
```
cflow [-r] [-d num] [-D name[=def]]... [-i incl] [-I dir]...
    [-U dir]... file...
```

DESCRIPTION

The *cflow* utility shall analyze a collection of object files or assembler, C-language, *lex*, or *yacc* source files, and attempt to build a graph, written to standard output, charting the external references.

OPTIONS

The *cflow* utility shall conform to XBD Section 12.2 (on page 215), except that the order of the **–D**, **–I**, and **–U** options (which are identical to their interpretation by *c17*) is significant.

The following options shall be supported:

–d *num* Indicate the depth at which the flowgraph is cut off. The application shall ensure that the argument *num* is a decimal integer. By default this is a very large number (typically greater than 32 000). Attempts to set the cut-off depth to a non-positive integer shall be ignored.

–i *incl* Increase the number of included symbols. The *incl* option-argument is one of the following characters:

x Include external and static data symbols. The default shall be to include only functions in the flowgraph.

_ (Underscore) Include names that begin with an <underscore>. The default shall be to exclude these functions (and data if **–i x** is used).

–r Reverse the caller:callee relationship, producing an inverted listing showing the callers of each function. The listing shall also be sorted in lexicographical order by callee.

OPERANDS

The following operand is supported:

file The pathname of a file for which a graph is to be generated. Filenames suffixed by **.l** shall shall be taken to be *lex* input, **.y** as *yacc* input, **.c** as *c17* input, and **.i** as the output of *c17* **–E**. Such files shall be processed as appropriate, determined by their suffix.

Files suffixed by **.s** (conventionally assembler source) may have more limited information extracted from them.

STDIN

Not used.

INPUT FILES

The input files shall be object files or assembler, C-language, *lex*, or *yacc* source files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *cflow*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

88953 *LC_ALL* If set to a non-empty string value, override the values of all the other
88954 internationalization variables.

88955 *LC_COLLATE*
88956 Determine the locale for the ordering of the output when the **–r** option is used.

88957 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
88958 characters (for example, single-byte as opposed to multi-byte characters in
88959 arguments and input files).

88960 *LC_MESSAGES*
88961 Determine the locale that should be used to affect the format and contents of
88962 diagnostic messages written to standard error.

88963 *NLSPATH* Determine the location of messages objects and message catalogs.

88964 **ASYNCHRONOUS EVENTS**
88965 Default.

88966 **STDOUT**
88967 The flowgraph written to standard output shall be formatted as follows:

88968 `"%d %s:%s\n", <reference number>, <global>, <definition>`

88969 Each line of output begins with a reference (that is, line) number, followed by indentation of at
88970 least one column position per level. This is followed by the name of the global, a <colon>, and
88971 its definition. Normally globals are only functions not defined as an external or beginning with
88972 an <underscore>; see the OPTIONS section for the **–i** inclusion option. For information extracted
88973 from C-language source, the definition consists of an abstract type declaration (for example, **char**
88974 ***) and, delimited by angle brackets, the name of the source file and the line number where the
88975 definition was found. Definitions extracted from object files indicate the filename and location
88976 counter under which the symbol appeared (for example, *text*).

88977 Once a definition of a name has been written, subsequent references to that name contain only
88978 the reference number of the line where the definition can be found. For undefined references,
88979 only `"<>"` shall be written.

88980 **STDERR**
88981 The standard error shall be used only for diagnostic messages.

88982 **OUTPUT FILES**
88983 None.

88984 **EXTENDED DESCRIPTION**
88985 None.

88986 **EXIT STATUS**
88987 The following exit values shall be returned:

88988 0 Successful completion.

88989 >0 An error occurred.

88990 **CONSEQUENCES OF ERRORS**
88991 Default.

APPLICATION USAGE

Files produced by *lex* and *yacc* cause the reordering of line number declarations, and this can confuse *cflow*. To obtain proper results, the input of *yacc* or *lex* must be directed to *cflow*.

EXAMPLES

Given the following in **file.c**:

```
int i;
int f();
int g();
int h();
int
main(void)
{
    f();
    g();
    f();
}
int
f()
{
    i = h();
}
```

The command:

```
cflow -i x file.c
```

produces the output:

```
1 main: int(), <file.c 6>
2    f: int(), <file.c 13>
3        h: <>
4        i: int, <file.c 1>
5    g: <>
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

c17, *lex*, *yacc*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

89034 **Issue 8**

89035 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

89036 Austin Group Defect 1195 is applied, changing ``main()'' to ``main(void)''.

NAME

chgrp — change the file group ownership

SYNOPSIS

```
chgrp [-h] group file...
chgrp -R [-H|-L|-P] group file...
```

DESCRIPTION

The *chgrp* utility shall set the group ID of the file named by each *file* operand to the group ID specified by the *group* operand.

For each *file* operand, or, if the **–R** option is used, each file encountered while walking the directory trees specified by the *file* operands, the *chgrp* utility shall perform actions equivalent to the *chown*() function defined in the System Interfaces volume of POSIX.1-2024, called with the following arguments:

- The *file* operand shall be used as the *path* argument.
- The user ID of the file shall be used as the *owner* argument.
- The specified group ID shall be used as the *group* argument.

Unless *chgrp* is invoked by a process with appropriate privileges, the set-user-ID and set-group-ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and set-group-ID bits of other file types may be cleared.

OPTIONS

The *chgrp* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported by the implementation:

–h For each *file* operand that names a file of type symbolic link, *chgrp* shall attempt to set the group ID of the symbolic link instead of the file referenced by the symbolic link.

–H If the **–R** option is specified and a symbolic link referencing a file of type directory is specified on the command line, *chgrp* shall change the group of the directory referenced by the symbolic link and all files in the file hierarchy below it.

–L If the **–R** option is specified and a symbolic link referencing a file of type directory is specified on the command line or encountered during the traversal of a file hierarchy, *chgrp* shall change the group of the directory referenced by the symbolic link and all files in the file hierarchy below it.

–P If the **–R** option is specified and a symbolic link is specified on the command line or encountered during the traversal of a file hierarchy, *chgrp* shall change the group ID of the symbolic link. The *chgrp* utility shall not follow the symbolic link to any other part of the file hierarchy.

–R Recursively change file group IDs. For each *file* operand that names a directory, *chgrp* shall change the group of the directory and all files in the file hierarchy below it. Unless a **–H**, **–L**, or **–P** option is specified, it is unspecified which of these options will be used as the default.

Specifying more than one of the mutually-exclusive options **–H**, **–L**, and **–P** shall not be considered an error. The last option specified shall determine the behavior of the utility.

89078 **OPERANDS**

89079 The following operands shall be supported:

89080 *group* A group name from the group database or a numeric group ID. Either specifies a
89081 group ID to be given to each file named by one of the *file* operands. If a numeric
89082 *group* operand exists in the group database as a group name, the group ID number
89083 associated with that group name is used as the group ID.

89084 *file* A pathname of a file whose group ID is to be modified.

89085 **STDIN**

89086 Not used.

89087 **INPUT FILES**

89088 None.

89089 **ENVIRONMENT VARIABLES**

89090 The following environment variables shall affect the execution of *chgrp*:

89091 *LANG* Provide a default value for the internationalization variables that are unset or null.
89092 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
89093 variables used to determine the values of locale categories.)

89094 *LC_ALL* If set to a non-empty string value, override the values of all the other
89095 internationalization variables.

89096 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
89097 characters (for example, single-byte as opposed to multi-byte characters in
89098 arguments).

89099 *LC_MESSAGES*
89100 Determine the locale that should be used to affect the format and contents of
89101 diagnostic messages written to standard error.

89102 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

89103 **ASYNCHRONOUS EVENTS**

89104 Default.

89105 **STDOUT**

89106 Not used.

89107 **STDERR**

89108 The standard error shall be used only for diagnostic messages.

89109 **OUTPUT FILES**

89110 None.

89111 **EXTENDED DESCRIPTION**

89112 None.

89113 **EXIT STATUS**

89114 The following exit values shall be returned:

89115 0 The utility executed successfully and all requested changes were made.

89116 >0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Only the owner of a file or the user with appropriate privileges may change the owner or group of a file.

Some implementations restrict the use of *chgrp* to a user with appropriate privileges when the *group* specified is not the effective group ID or one of the supplementary group IDs of the calling process.

EXAMPLES

None.

RATIONALE

The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it can overflow the range of valid exit status values. The standard developers chose to mask these by specifying only 0 and >0 as exit values.

The functionality of *chgrp* is described substantially through references to *chown*( ). In this way, there is no duplication of effort required for describing the interactions of permissions, multiple groups, and so on.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod, *chown*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *chown*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

New options –**H**, –**L**, and –**P** are added to align with the IEEE P1003.2b draft standard. These options affect the processing of symbolic links.

IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS section to ``Default.''.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/15 is applied, changing the SYNOPSIS to make it clear that –**h** and –**R** are optional.

Issue 7

SD5-XCU-ERN-8 is applied, removing the –**R** from the first line of the SYNOPSIS.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0080 [237,341] is applied.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

89156 **NAME**

89157 chmod — change the file modes

89158 **SYNOPSIS**

89159 `chmod [-R] mode file...`

89160 **DESCRIPTION**

89161 The *chmod* utility shall change any or all of the file mode bits of the file named by each *file*
89162 operand in the way specified by the *mode* operand.

89163 It is implementation-defined whether and how the *chmod* utility affects any alternate or
89164 additional file access control mechanism (see XBD Section 4.7, on page 97) being used for the
89165 specified file.

89166 Only a process whose effective user ID matches the user ID of the file, or a process with
89167 appropriate privileges, shall be permitted to change the file mode bits of a file.

89168 Upon successfully changing the file mode bits of a file, the *chmod* utility shall mark for update
89169 the last file status change timestamp of the file.

89170 **OPTIONS**

89171 The *chmod* utility shall conform to XBD Section 12.2 (on page 215).

89172 The following option shall be supported:

89173 **–R** Recursively change file mode bits. For each *file* operand that names a directory,
89174 *chmod* shall change the file mode bits of the directory and all files in the file
89175 hierarchy below it.

89176 **OPERANDS**

89177 The following operands shall be supported:

89178 *mode* Represents the change to be made to the file mode bits of each file named by one of
89179 the *file* operands; see the EXTENDED DESCRIPTION section.

89180 *file* A pathname of a file whose file mode bits shall be modified.

89181 **STDIN**

89182 Not used.

89183 **INPUT FILES**

89184 None.

89185 **ENVIRONMENT VARIABLES**

89186 The following environment variables shall affect the execution of *chmod*:

89187 *LANG* Provide a default value for the internationalization variables that are unset or null.
89188 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
89189 variables used to determine the values of locale categories.)

89190 *LC_ALL* If set to a non-empty string value, override the values of all the other
89191 internationalization variables.

89192 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
89193 characters (for example, single-byte as opposed to multi-byte characters in
89194 arguments).

89195 *LC_MESSAGES*

89196 Determine the locale that should be used to affect the format and contents of
89197 diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

The *mode* operand shall be either a *symbolic_mode* expression or a non-negative octal integer. The *symbolic_mode* form is described by the grammar later in this section.

Each **clause** shall specify an operation to be performed on the current file mode bits of each *file*. The operations shall be performed on each *file* in the order in which the **clause**s are specified.

The **who** symbols **u**, **g**, and **o** shall specify the *user*, *group*, and *other* parts of the file mode bits, respectively. A **who** consisting of the symbol **a** shall be equivalent to **ugo**.

The **perm** symbols **r**, **w**, and **x** represent the *read*, *write*, and *execute/search* portions of file mode bits, respectively. The **perm** symbol **s** shall represent the *set-user-ID-on-execution* (when **who** contains or implies **u**) and *set-group-ID-on-execution* (when **who** contains or implies **g**) bits.

The **perm** symbol **X** shall represent the execute/search portion of the file mode bits if the file is a directory or if the current (unmodified) file mode bits have at least one of the execute bits (S_IXUSR, S_IXGRP, or S_IXOTH) set. It shall be ignored if the file is not a directory and none of the execute bits are set in the current file mode bits.

The **permcopy** symbols **u**, **g**, and **o** shall represent the current permissions associated with the user, group, and other parts of the file mode bits, respectively. For the remainder of this section, **perm** refers to the non-terminals **perm** and **permcopy** in the grammar.

If multiple **actionlist**s are grouped with a single **wholist** in the grammar, each **actionlist** shall be applied in the order specified with that **wholist**. The *op* symbols shall represent the operation performed, as follows:

\+ If **perm** is not specified, the `'+'` operation shall not change the file mode bits.

If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and other permissions, except for those with corresponding bits in the file mode creation mask of the invoking process, shall be set.

Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.

– If **perm** is not specified, the `'-'` operation shall not change the file mode bits.

If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and other permissions, except for those with corresponding bits in the file mode creation mask of the invoking process, shall be cleared.

Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be cleared.

= Clear the file mode bits specified by the **who** value, or, if no **who** value is specified, all of the file mode bits specified in this volume of POSIX.1-2024.

89240 If **perm** is not specified, the `'='` operation shall make no further modifications to the file
89241 mode bits.

89242 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and
89243 other permissions, except for those with corresponding bits in the file mode creation mask
89244 of the invoking process, shall be set.

89245 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.

89246 When using the symbolic mode form on a regular file, it is implementation-defined whether or
89247 not:

89248 • Requests to set the set-user-ID-on-execution or set-group-ID-on-execution bit when all
89249 execute bits are currently clear and none are being set are ignored.

89250 • Requests to clear all execute bits also clear the set-user-ID-on-execution and set-group-ID-
89251 on-execution bits.

89252 • Requests to clear the set-user-ID-on-execution or set-group-ID-on-execution bits when all
89253 execute bits are currently clear are ignored. However, if the command *ls* **–l** *file* writes an *s*
89254 in the position indicating that the set-user-ID-on-execution or set-group-ID-on-execution is
89255 set, the commands *chmod* **u–s** *file* or *chmod* **g–s** *file*, respectively, shall not be ignored.

89256 When using the symbolic mode form on other file types, it is implementation-defined whether
89257 or not requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are
89258 honored.

89259 If the **who** symbol **o** is used in conjunction with the **perm** symbol **s** with no other **who** symbols
89260 being specified, the set-user-ID-on-execution and set-group-ID-on-execution bits shall not be
89261 modified. It shall not be an error to specify the **who** symbol **o** in conjunction with the **perm**
89262 symbol **s**.

89263 XSI The **perm** symbol **t** shall specify the S_ISVTX bit. When used with a file of type directory, it can
89264 be used with the **who** symbol **a**, or with no **who** symbol. It shall not be an error to specify a **who**
89265 symbol of **u**, **g**, or **o** in conjunction with the **perm** symbol **t**, but the meaning of these
89266 combinations is unspecified. The effect when using the **perm** symbol **t** with any file type other
89267 than directory is unspecified.

89268 For an octal integer *mode* operand, the file mode bits shall be set absolutely.

89269 For each bit set in the octal number, the corresponding file permission bit shown in the following
89270 table shall be set; all other file permission bits shall be cleared. For regular files, for each bit set in
89271 the octal number corresponding to the set-user-ID-on-execution or the set-group-ID-on-
89272 execution, bits shown in the following table shall be set; if these bits are not set in the octal
89273 number, they are cleared. For other file types, it is implementation-defined whether or not
89274 requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are
89275 honored.

89276

Octal	Mode Bit	Octal	Mode Bit	Octal	Mode Bit	Octal	Mode Bit
4000	S_ISUID	**0400**	S_IRUSR	**0040**	S_IRGRP	**0004**	S_IROTH
2000	S_ISGID	**0200**	S_IWUSR	**0020**	S_IWGRP	**0002**	S_IWOTH
XSI **1000**	S_ISVTX	**0100**	S_IXUSR	**0010**	S_IXGRP	**0001**	S_IXOTH

89280 When bits are set in the octal number other than those listed in the table above, the behavior is
89281 unspecified.

Grammar for chmod

The grammar and lexical conventions in this section describe the syntax for the *symbolic_mode* operand. The general conventions for this style of grammar are described in Section 1.3 (on page 2461). A valid *symbolic_mode* can be represented as the non-terminal symbol *symbolic_mode* in the grammar. This formal syntax shall take precedence over the preceding text syntax description.

The lexical processing is based entirely on single characters. Implementations need not allow <blank> characters within the single argument being processed.

```
%start    symbolic_mode
%%

symbolic_mode    : clause
                 | symbolic_mode ',' clause
                 ;

clause           : actionlist
                 | wholist actionlist
                 ;

wholist          : who
                 | wholist who
                 ;

who              : 'u' | 'g' | 'o' | 'a'
                 ;

actionlist       : action
                 | actionlist action
                 ;

action           : op
                 | op permlist
                 | op permcopy
                 ;

permcopy         : 'u' | 'g' | 'o'
                 ;

op               : '+' | '-' | '='
                 ;

permlist         : perm
                 | perm permlist
                 ;

XSI  perm        : 'r' | 'w' | 'x' | 'X' | 's' | 't'
                 ;
```

EXIT STATUS

The following exit values shall be returned:

0 The utility executed successfully and all requested changes were made.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Some implementations of the *chmod* utility change the mode of a directory before the files in the directory when performing a recursive (**–R** option) change; others change the directory mode after the files in the directory. If an application tries to remove read or search permission for a file hierarchy, the removal attempt fails if the directory is changed first; on the other hand, trying to re-enable permissions to a restricted hierarchy fails if directories are changed last. Users should not try to make a hierarchy inaccessible to themselves.

Some implementations of *chmod* never used the *umask* of the process when changing modes; systems conformant with this volume of POSIX.1-2024 do so when **who** is not specified. Note the difference between:

```
chmod a-w file
```

which removes all write permissions, and:

```
chmod -- -w file
```

which removes write permissions that would be allowed if **file** was created with the same *umask*.

Conforming applications should never assume that they know how the set-user-ID and set-group-ID bits on directories are interpreted.

EXAMPLES

Mode	Results
a+=	Equivalent to *a*+,*a*=; clears all file mode bits.
go+–w	Equivalent to *go*+,*go*–*w*; clears group and other write bits.
g=*o*–*w*	Equivalent to *g*=*o*,*g*–*w*; sets group bit to match other bits and then clears group write bit.
g–*r*+*w*	Equivalent to *g*–*r*,*g*+*w*; clears group read bit and sets group write bit.
uo=*g*	Sets owner bits to match group bits and sets other bits to match group bits.

RATIONALE

The functionality of *chmod* is described substantially through references to concepts defined in the System Interfaces volume of POSIX.1-2024. In this way, there is less duplication of effort required for describing the interactions of permissions. However, the behavior of this utility is not described in terms of the *chmod*( ) function from the System Interfaces volume of POSIX.1-2024 because that specification requires certain side-effects upon alternate file access control mechanisms that might not be appropriate, depending on the implementation.

Implementations that support mandatory file and record locking as specified by the 1984 /usr/group standard historically used the combination of set-group-ID bit set and group execute bit clear to indicate mandatory locking. This condition is usually set or cleared with the symbolic mode **perm** symbol **l** instead of the **perm** symbols **s** and **x** so that the mandatory locking mode is not changed without explicit indication that that was what the user intended. Therefore, the details on how the implementation treats these conditions must be defined in the documentation. This volume of POSIX.1-2024 does not require mandatory locking (nor does the System Interfaces volume of POSIX.1-2024), but does allow it as an extension. However, this volume of POSIX.1-2024 does require that the *ls* and *chmod* utilities work consistently in this

area. If *ls* **–l** *file* indicates that the set-group-ID bit is set, *chmod* **g–s** *file* must clear it (assuming appropriate privileges exist to change modes).

The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it can overflow the range of valid exit status values. This problem is avoided here by specifying only 0 and >0 as exit values.

The System Interfaces volume of POSIX.1-2024 indicates that implementation-defined restrictions may cause the S_ISUID and S_ISGID bits to be ignored. This volume of POSIX.1-2024 allows the *chmod* utility to choose to modify these bits before calling *chmod*( ) (or some function providing equivalent capabilities) for non-regular files. Among other things, this allows implementations that use the set-user-ID and set-group-ID bits on directories to enable extended features to handle these extensions in an intelligent manner.

The **X perm** symbol was adopted from BSD-based systems because it provides commonly desired functionality when doing recursive (**–R** option) modifications. Similar functionality is not provided by the *find* utility. Historical BSD versions of *chmod*, however, only supported **X** with *op*+; it has been extended in this volume of POSIX.1-2024 because it is also useful with *op*=. (It has also been added for *op*– even though it duplicates **x**, in this case, because it is intuitive and easier to explain.)

The grammar was extended with the *permcopy* non-terminal to allow historical-practice forms of symbolic modes like **o=u –g** (that is, set the ``other'' permissions to the permissions of ``owner'' minus the permissions of ``group'').

FUTURE DIRECTIONS

None.

SEE ALSO

ls, *umask*

XBD Section 4.7 (on page 97), Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *chmod*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- Octal modes have been kept and made mandatory despite being marked obsolescent in the ISO POSIX-2: 1993 standard.

IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS section to ``Default.''.

The Open Group Base Resolution bwg2001-010 is applied, adding the description of the S_ISVTX bit and the **t perm** symbol as part of the XSI option.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/16 is applied, changing the XSI shaded text in the EXTENDED DESCRIPTION from:

> ``The **perm** symbol **t** shall specify the S_ISVTX bit and shall apply to directories only. The effect when using it with any other file type is unspecified. It can be used with the **who** symbols **o**, **a**, or with no **who** symbol. It shall not be an error to specify a **who** symbol of **u** or **g** in conjunction with the **perm** symbol **t**; it shall be ignored for **u** and **g**.''

to:

> ``The **perm** symbol **t** shall specify the S_ISVTX bit. When used with a file of type directory, it can be used with the **who** symbol **a**, or with no **who** symbol. It shall not be an error to specify a **who** symbol of **u**, **g**, or **o** in conjunction with the **perm** symbol **t**, but the meaning of these combinations is unspecified. The effect when using the **perm** symbol **t** with any file type other than directory is unspecified.''

This change is to permit historical behavior.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Austin Group Interpretation 1003.1-2001 #130 is applied, adding text to the DESCRIPTION about about marking for update the last file status change timestamp of the file.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

89426 **NAME**

89427 chown — change the file ownership

89428 **SYNOPSIS**

89429 `chown [-h] owner[:group] file...`

89430 `chown -R [-H|-L|-P] owner[:group] file...`

89431 **DESCRIPTION**

89432 The *chown* utility shall set the user ID of the file named by each *file* operand to the user ID
89433 specified by the *owner* operand.

89434 For each *file* operand, or, if the **-R** option is used, each file encountered while walking the
89435 directory trees specified by the *file* operands, the *chown* utility shall perform actions equivalent to
89436 the *chown*() function defined in the System Interfaces volume of POSIX.1-2024, called with the
89437 following arguments:

89438 1. The *file* operand shall be used as the *path* argument.

89439 2. The user ID indicated by the *owner* portion of the first operand shall be used as the *owner*
89440 argument.

89441 3. If the *group* portion of the first operand is given, the group ID indicated by it shall be used
89442 as the *group* argument; otherwise, the group ownership shall not be changed.

89443 Unless *chown* is invoked by a process with appropriate privileges, the set-user-ID and set-group-
89444 ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and set-
89445 group-ID bits of other file types may be cleared.

89446 **OPTIONS**

89447 The *chown* utility shall conform to XBD Section 12.2 (on page 215).

89448 The following options shall be supported by the implementation:

89449 **-h** For each file operand that names a file of type symbolic link, *chown* shall attempt to
89450 set the user ID of the symbolic link. If a group ID was specified, for each file
89451 operand that names a file of type symbolic link, *chown* shall attempt to set the
89452 group ID of the symbolic link.

89453 **-H** If the **-R** option is specified and a symbolic link referencing a file of type directory
89454 is specified on the command line, *chown* shall change the user ID (and group ID, if
89455 specified) of the directory referenced by the symbolic link and all files in the file
89456 hierarchy below it.

89457 **-L** If the **-R** option is specified and a symbolic link referencing a file of type directory
89458 is specified on the command line or encountered during the traversal of a file
89459 hierarchy, *chown* shall change the user ID (and group ID, if specified) of the
89460 directory referenced by the symbolic link and all files in the file hierarchy below it.

89461 **-P** If the **-R** option is specified and a symbolic link is specified on the command line
89462 or encountered during the traversal of a file hierarchy, *chown* shall change the
89463 owner ID (and group ID, if specified) of the symbolic link. The *chown* utility shall
89464 not follow the symbolic link to any other part of the file hierarchy.

89465 **-R** Recursively change file user and group IDs. For each *file* operand that names a
89466 directory, *chown* shall change the user ID (and group ID, if specified) of the
89467 directory and all files in the file hierarchy below it. Unless a **-H**, **-L**, or **-P** option is
89468 specified, it is unspecified which of these options will be used as the default.

89469 Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be

considered an error. The last option specified shall determine the behavior of the utility.

OPERANDS

The following operands shall be supported:

owner[:*group*] A user ID and optional group ID to be assigned to *file*. The *owner* portion of this operand shall be a user name from the user database or a numeric user ID. Either specifies a user ID which shall be given to each file named by one of the *file* operands. If a numeric *owner* operand exists in the user database as a user name, the user ID number associated with that user name shall be used as the user ID. Similarly, if the *group* portion of this operand is present, it shall be a group name from the group database or a numeric group ID. Either specifies a group ID which shall be given to each file. If a numeric group operand exists in the group database as a group name, the group ID number associated with that group name shall be used as the group ID.

file A pathname of a file whose user ID is to be modified.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *chown*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

89510 **EXTENDED DESCRIPTION**

89511 None.

89512 **EXIT STATUS**

89513 The following exit values shall be returned:

89514 0 The utility executed successfully and all requested changes were made.

89515 >0 An error occurred.

89516 **CONSEQUENCES OF ERRORS**

89517 Default.

89518 **APPLICATION USAGE**

89519 Only the owner of a file or the user with appropriate privileges may change the owner or group
89520 of a file.

89521 Some implementations restrict the use of *chown* to a user with appropriate privileges.

89522 **EXAMPLES**

89523 None.

89524 **RATIONALE**

89525 The System V and BSD versions use different exit status codes. Some implementations used the
89526 exit status as a count of the number of errors that occurred; this practice is unworkable since it
89527 can overflow the range of valid exit status values. These are masked by specifying only 0 and >0
89528 as exit values.

89529 The functionality of *chown* is described substantially through references to functions in the
89530 System Interfaces volume of POSIX.1-2024. In this way, there is no duplication of effort required
89531 for describing the interactions of permissions, multiple groups, and so on.

89532 The 4.3 BSD method of specifying both owner and group was included in this volume of
89533 POSIX.1-2024 because:

89534 • There are cases where the desired end condition could not be achieved using the *chgrp* and
89535 *chown* (that only changed the user ID) utilities. (If the current owner is not a member of the
89536 desired group and the desired owner is not a member of the current group, the *chown*( )
89537 function could fail unless both owner and group are changed at the same time.)

89538 • Even if they could be changed independently, in cases where both are being changed, there
89539 is a 100% performance penalty caused by being forced to invoke both utilities.

89540 The BSD syntax *user*[.*group*] was changed to *user*[:*group*] in this volume of POSIX.1-2024 because
89541 the <period> is a valid character in login names (as specified by the Base Definitions volume of
89542 POSIX.1-2024, login names consist of characters in the portable filename character set). The
89543 <colon> character was chosen as the replacement for the <period> character because it would
89544 never be allowed as a character in a user name or group name on historical implementations.

89545 The **−R** option is considered by some observers as an undesirable departure from the historical
89546 UNIX system tools approach; since a tool, *find*, already exists to recurse over directories, there
89547 seemed to be no good reason to require other tools to have to duplicate that functionality.
89548 However, the **−R** option was deemed an important user convenience, is far more efficient than
89549 forking a separate process for each element of the directory hierarchy, and is in widespread
89550 historical use.

FUTURE DIRECTIONS

None.

SEE ALSO

chgrp, *chmod*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *chown*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

New options **–h**, **–H**, **–L**, and **–P** are added to align with the IEEE P1003.2b draft standard. These options affect the processing of symbolic links.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS section to ``Default.''.

The ``otherwise, ...'' text in item 3. of the DESCRIPTION is changed to ``otherwise, the group ownership shall not be changed''.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/17 is applied, changing the SYNOPSIS to make it clear that **–h** and **–R** are optional.

Issue 7

SD5-XCU-ERN-9 is applied, removing the **–R** from the first line of the SYNOPSIS.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The description of the **–h** and **–P** options is revised.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

89575 **NAME**

89576 cksum — write file checksums and sizes

89577 **SYNOPSIS**

89578 `cksum [`*`file...`*`]`

89579 **DESCRIPTION**

89580 The *cksum* utility shall calculate and write to standard output a cyclic redundancy check (CRC)
89581 for each input file, and also write to standard output the number of octets in each file. The CRC
89582 used is based on the polynomial used for CRC error checking in the ISO/IEC 8802-3:1996
89583 standard (Ethernet).

89584 The encoding for the CRC checksum is defined by the generating polynomial:

89585 $G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

89586 Mathematically, the CRC value corresponding to a given file shall be defined by the following
89587 procedure:

89588 1. The n bits to be evaluated are considered to be the coefficients of a mod 2 polynomial
89589 $M(x)$ of degree $n-1$. These n bits are the bits from the file, with the most significant bit
89590 being the most significant bit of the first octet of the file and the last bit being the least
89591 significant bit of the last octet, padded with zero bits (if necessary) to achieve an integral
89592 number of octets, followed by one or more octets representing the length of the file as a
89593 binary value, least significant octet first. The smallest number of octets capable of
89594 representing this integer shall be used.

89595 2. $M(x)$ is multiplied by x^{32} (that is, shifted left 32 bits) and divided by $G(x)$ using mod 2
89596 division, producing a remainder $R(x)$ of degree ≤ 31.

89597 3. The coefficients of $R(x)$ are considered to be a 32-bit sequence.

89598 4. The bit sequence is complemented and the result is the CRC.

89599 **OPTIONS**

89600 None.

89601 **OPERANDS**

89602 The following operand shall be supported:

89603 *file* A pathname of a file to be checked. If no *file* operands are specified, the standard
89604 input shall be used.

89605 **STDIN**

89606 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*
89607 operand is `'-'` and the implementation treats the `'-'` as meaning standard input. Otherwise,
89608 the standard input shall not be used. See the INPUT FILES section.

89609 **INPUT FILES**

89610 The input files can be any file type.

89611 **ENVIRONMENT VARIABLES**

89612 The following environment variables shall affect the execution of *cksum*:

89613 *LANG* Provide a default value for the internationalization variables that are unset or null.
89614 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
89615 variables used to determine the values of locale categories.)

89616 *LC_ALL* If set to a non-empty string value, override the values of all the other
89617 internationalization variables.

89618 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
89619 characters (for example, single-byte as opposed to multi-byte characters in
89620 arguments).

89621 *LC_MESSAGES*
89622 Determine the locale that should be used to affect the format and contents of
89623 diagnostic messages written to standard error.

89624 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

89625 **ASYNCHRONOUS EVENTS**
89626 Default.

89627 **STDOUT**
89628 For each file processed successfully, the *cksum* utility shall write in the following format:

89629 `"%u %d %s\n", <checksum>, <# of octets>, <pathname>`

89630 If no *file* operand was specified, the pathname and its leading <space> shall be omitted.

89631 **STDERR**
89632 The standard error shall be used only for diagnostic messages.

89633 **OUTPUT FILES**
89634 None.

89635 **EXTENDED DESCRIPTION**
89636 None.

89637 **EXIT STATUS**
89638 The following exit values shall be returned:

89639 0 All files were processed successfully.

89640 >0 An error occurred.

89641 **CONSEQUENCES OF ERRORS**
89642 Default.

89643 **APPLICATION USAGE**
89644 The *cksum* utility is typically used to quickly compare a suspect file against a trusted version of
89645 the same, such as to ensure that files transmitted over noisy media arrive intact. However, this
89646 comparison cannot be considered cryptographically secure. This utility should be avoided
89647 whenever non-trivial requirements (including safety and security) have to be fulfilled.

89648 Although input files to *cksum* can be any type, the results need not be what would be expected
89649 on character special device files or on file types not described by the System Interfaces volume of
89650 POSIX.1-2024. Since this volume of POSIX.1-2024 does not specify the block size used when
89651 doing input, checksums of character special files need not process all of the data in those files.

89652 The algorithm is expressed in terms of a bitstream divided into octets. If a file is transmitted
89653 between two systems and undergoes any data transformation (such as changing little-endian
89654 byte ordering to big-endian), identical CRC values cannot be expected. Implementations
89655 performing such transformations may extend *cksum* to handle such situations.

89656 **EXAMPLES**
89657 None.

RATIONALE

The *cksum* utility is included in this standard for reasons of portability but is not suitable for uses where non-trivial requirements (including safety and security) have to be fulfilled. Implementations are encouraged to provide utilities that implement hash and integrity checksum algorithms of higher security and to keep up to date with developments in this area.

The following C-language program can be used as a model to describe the algorithm. It assumes that a **char** is one octet. It also assumes that the entire file is available for one pass through the function. This was done for simplicity in demonstrating the algorithm, rather than as an implementation model.

```
static unsigned long crctab[] = {
0x00000000,
0x04c11db7, 0x09823b6e, 0x0d4326d9, 0x130476dc, 0x17c56b6b,
0x1a864db2, 0x1e475005, 0x2608edb8, 0x22c9f00f, 0x2f8ad6d6,
0x2b4bcb61, 0x350c9b64, 0x31cd86d3, 0x3c8ea00a, 0x384fbdbd,
0x4c11db70, 0x48d0c6c7, 0x4593e01e, 0x4152fda9, 0x5f15adac,
0x5bd4b01b, 0x569796c2, 0x52568b75, 0x6a1936c8, 0x6ed82b7f,
0x639b0da6, 0x675a1011, 0x791d4014, 0x7ddc5da3, 0x709f7b7a,
0x745e66cd, 0x9823b6e0, 0x9ce2ab57, 0x91a18d8e, 0x95609039,
0x8b27c03c, 0x8fe6dd8b, 0x82a5fb52, 0x8664e6e5, 0xbe2b5b58,
0xbaea46ef, 0xb7a96036, 0xb3687d81, 0xad2f2d84, 0xa9ee3033,
0xa4ad16ea, 0xa06c0b5d, 0xd4326d90, 0xd0f37027, 0xddb056fe,
0xd9714b49, 0xc7361b4c, 0xc3f706fb, 0xceb42022, 0xca753d95,
0xf23a8028, 0xf6fb9d9f, 0xfbb8bb46, 0xff79a6f1, 0xe13ef6f4,
0xe5ffeb43, 0xe8bccd9a, 0xec7dd02d, 0x34867077, 0x30476dc0,
0x3d044b19, 0x39c556ae, 0x278206ab, 0x23431b1c, 0x2e003dc5,
0x2ac12072, 0x128e9dcf, 0x164f8078, 0x1b0ca6a1, 0x1fcdbb16,
0x018aeb13, 0x054bf6a4, 0x0808d07d, 0x0cc9cdca, 0x7897ab07,
0x7c56b6b0, 0x71159069, 0x75d48dde, 0x6b93dddb, 0x6f52c06c,
0x6211e6b5, 0x66d0fb02, 0x5e9f46bf, 0x5a5e5b08, 0x571d7dd1,
0x53dc6066, 0x4d9b3063, 0x495a2dd4, 0x44190b0d, 0x40d816ba,
0xaca5c697, 0xa864db20, 0xa527fdf9, 0xa1e6e04e, 0xbfa1b04b,
0xbb60adfc, 0xb6238b25, 0xb2e29692, 0x8aad2b2f, 0x8e6c3698,
0x832f1041, 0x87ee0df6, 0x99a95df3, 0x9d684044, 0x902b669d,
0x94ea7b2a, 0xe0b41de7, 0xe4750050, 0xe9362689, 0xedf73b3e,
0xf3b06b3b, 0xf771768c, 0xfa325055, 0xfef34de2, 0xc6bcf05f,
0xc27dede8, 0xcf3ecb31, 0xcbffd686, 0xd5b88683, 0xd1799b34,
0xdc3abded, 0xd8fba05a, 0x690ce0ee, 0x6dcdfd59, 0x608edb80,
0x644fc637, 0x7a089632, 0x7ec98b85, 0x738aad5c, 0x774bb0eb,
0x4f040d56, 0x4bc510e1, 0x46863638, 0x42472b8f, 0x5c007b8a,
0x58c1663d, 0x558240e4, 0x51435d53, 0x251d3b9e, 0x21dc2629,
0x2c9f00f0, 0x285e1d47, 0x36194d42, 0x32d850f5, 0x3f9b762c,
0x3b5a6b9b, 0x0315d626, 0x07d4cb91, 0x0a97ed48, 0x0e56f0ff,
0x1011a0fa, 0x14d0bd4d, 0x19939b94, 0x1d528623, 0xf12f560e,
0xf5ee4bb9, 0xf8ad6d60, 0xfc6c70d7, 0xe22b20d2, 0xe6ea3d65,
0xeba91bbc, 0xef68060b, 0xd727bbb6, 0xd3e6a601, 0xdea580d8,
0xda649d6f, 0xc423cd6a, 0xc0e2d0dd, 0xcda1f604, 0xc960ebb3,
0xbd3e8d7e, 0xb9ff90c9, 0xb4bcb610, 0xb07daba7, 0xae3afba2,
0xaafbe615, 0xa7b8c0cc, 0xa379dd7b, 0x9b3660c6, 0x9ff77d71,
0x92b45ba8, 0x9675461f, 0x8832161a, 0x8cf30bad, 0x81b02d74,
0x857130c3, 0x5d8a9099, 0x594b8d2e, 0x5408abf7, 0x50c9b640,
0x4e8ee645, 0x4a4ffbf2, 0x470cdd2b, 0x43cdc09c, 0x7b827d21,
```



```
0x7f436096, 0x7200464f, 0x76c15bf8, 0x68860bfd, 0x6c47164a,
0x61043093, 0x65c52d24, 0x119b4be9, 0x155a565e, 0x18197087,
0x1cd86d30, 0x029f3d35, 0x065e2082, 0x0b1d065b, 0x0fdc1bec,
0x3793a651, 0x3352bbe6, 0x3e119d3f, 0x3ad08088, 0x2497d08d,
0x2056cd3a, 0x2d15ebe3, 0x29d4f654, 0xc5a92679, 0xc1683bce,
0xcc2b1d17, 0xc8ea00a0, 0xd6ad50a5, 0xd26c4d12, 0xdf2f6bcb,
0xdbee767c, 0xe3a1cbc1, 0xe760d676, 0xea23f0af, 0xeee2ed18,
0xf0a5bd1d, 0xf464a0aa, 0xf9278673, 0xfde69bc4, 0x89b8fd09,
0x8d79e0be, 0x803ac667, 0x84fbdbd0, 0x9abc8bd5, 0x9e7d9662,
0x933eb0bb, 0x97ffad0c, 0xafb010b1, 0xab710d06, 0xa6322bdf,
0xa2f33668, 0xbcb4666d, 0xb8757bda, 0xb5365d03, 0xb1f740b4
};

unsigned long memcrc(const unsigned char *b, size_t n)
{
/*  Input arguments:
 *  const unsigned char*   b == byte sequence to checksum
 *  size_t                 n == length of sequence
 */

    register size_t i;
    register unsigned c, s = 0;

    for (i = n; i > 0; --i) {
        c = *b++;
        s = (s << 8) ^ crctab[(s >> 24) ^ c];
    }

    /* Extend with the length of the string. */
    while (n != 0) {
        c = n & 0377;
        n >>= 8;
        s = (s << 8) ^ crctab[(s >> 24) ^ c];
    }

    return ~s;
}
```

The historical practice of writing the number of ``blocks'' has been changed to writing the number of octets, since the latter is not only more useful, but also since historical implementations have not been consistent in defining what a ``block'' meant.

The algorithm used was selected to increase the operational robustness of *cksum*. Neither the System V nor BSD *sum* algorithm was selected. Since each of these was different and each was the default behavior on those systems, no realistic compromise was available if either were selected—some set of historical applications would break. Therefore, the name was changed to *cksum*. Although the historical *sum* commands will probably continue to be provided for many years, programs designed for portability across systems should use the new name.

The algorithm selected is based on that used by the ISO/IEC 8802-3: 1996 standard (Ethernet) for the frame check sequence field. The algorithm used does not match the technical definition of a *checksum*; the term is used for historical reasons. The length of the file is included in the CRC calculation because this parallels inclusion of a length field by Ethernet in its CRC, but also because it guards against inadvertent collisions between files that begin with different series of zero octets. The chance that two different files produce identical CRCs is much greater when their lengths are not considered. Keeping the length and the checksum of the file itself separate

would yield a slightly more robust algorithm, but historical usage has always been that a single number (the checksum as printed) represents the signature of the file. It was decided that historical usage was the more important consideration.

Early proposals contained modifications to the Ethernet algorithm that involved extracting table values whenever an intermediate result became zero. This was demonstrated to be less robust than the current method and mathematically difficult to describe or justify.

The calculation used is identical to that given in pseudo-code in the referenced Sarwate article. The pseudo-code rendition is:

```
X <- 0; Y <- 0;
for i <- m -1 step -1 until 0 do
    begin
    T <- X(1) ^ A[i];
    X(1) <- X(0); X(0) <- Y(1); Y(1) <- Y(0); Y(0) <- 0;
    comment: f[T] and f'[T] denote the T-th words in the
        table f and f' ;
    X <- X ^ f[T]; Y <- Y ^ f'[T];
    end
```

The pseudo-code is reproduced exactly as given; however, note that in the case of *cksum*, **A[i]** represents a byte of the file, the words **X** and **Y** are treated as a single 32-bit value, and the tables **f** and **f′** are a single table containing 32-bit values.

The referenced Sarwate article also discusses generating the table.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

XBD Chapter 8 (on page 167)

CHANGE HISTORY

First released in Issue 4.

Issue 7

Austin Group Interpretation 1003.1-2001 #092 is applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0081 [446] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1041 is applied, changing the APPLICATION USAGE and RATIONALE sections.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

89799 **NAME**

89800 cmp — compare two files

89801 **SYNOPSIS**

89802 `cmp [-l|-s] file1 file2`

89803 **DESCRIPTION**

89804 The *cmp* utility shall compare two files. The *cmp* utility shall write no output if the files are the
89805 same. Under default options, if they differ, it shall write to standard output the byte and line
89806 number at which the first difference occurred. Bytes and lines shall be numbered beginning with
89807 1.

89808 **OPTIONS**

89809 The *cmp* utility shall conform to XBD Section 12.2 (on page 215).

89810 The following options shall be supported:

89811 **–l** (Lowercase ell.) Write the byte number (decimal) and the differing bytes (octal) for
89812 each difference.

89813 **–s** Write nothing to standard output or standard error when files differ; indicate
89814 differing files through exit status only. It is unspecified whether a diagnostic
89815 message is written to standard error when an error is encountered; if a message is
89816 not written, the error is indicated through exit status only.

89817 **OPERANDS**

89818 The following operands shall be supported:

89819 *file1* A pathname of the first file to be compared. If *file1* is `'-'`, the standard input shall
89820 be used.

89821 *file2* A pathname of the second file to be compared. If *file2* is `'-'`, the standard input
89822 shall be used.

89823 If both *file1* and *file2* refer to standard input or refer to the same FIFO special, block special, or
89824 character special file, the results are undefined.

89825 **STDIN**

89826 The standard input shall be used only if the *file1* or *file2* operand refers to standard input. See the
89827 INPUT FILES section.

89828 **INPUT FILES**

89829 The input files can be any file type.

89830 **ENVIRONMENT VARIABLES**

89831 The following environment variables shall affect the execution of *cmp*:

89832 *LANG* Provide a default value for the internationalization variables that are unset or null.
89833 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
89834 variables used to determine the values of locale categories.)

89835 *LC_ALL* If set to a non-empty string value, override the values of all the other
89836 internationalization variables.

89837 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
89838 characters (for example, single-byte as opposed to multi-byte characters in
89839 arguments).

89840 *LC_MESSAGES*

89841 Determine the locale that should be used to affect the format and contents of
89842 diagnostic messages written to standard error and informative messages written to

standard output.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

In the POSIX locale, results of the comparison shall be written to standard output. When no options are used, the format shall be:

```
"%s %s differ: char %d, line %d\n", file1, file2,
    <byte number>, <line number>
```

When the **–l** option is used, the format shall be:

```
"%d %o %o\n", <byte number>, <differing byte>,
    <differing byte>
```

for each byte that differs. The first **<***differing byte***>** number is from *file1* while the second is from *file2*. In both cases, **<***byte number***>** shall be relative to the beginning of the file, beginning with 1.

No output shall be written to standard output when the **–s** option is used.

STDERR

The standard error shall be used only for diagnostic messages. If the **–l** option is used and *file1* and *file2* differ in length, or if the **–s** option is not used and *file1* and *file2* are identical for the entire length of the shorter file, in the POSIX locale the following diagnostic message shall be written:

```
"cmp: EOF on %s%s\n", <name of shorter file>, <additional info>
```

The **<***additional info***>** field shall either be null or a string that starts with a **<**blank**>** and contains no **<**newline**>** characters. Some implementations report on the number of lines in this case.

If the **–s** option is used and an error occurs, it is unspecified whether a diagnostic message is written to standard error.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 The files are identical.

1 The files are different; this includes the case where one file is identical to the first part of the other.

>1 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Although input files to *cmp* can be any type, the results might not be what would be expected on character special device files or on file types not described by the System Interfaces volume of POSIX.1-2024. Since this volume of POSIX.1-2024 does not specify the block size used when doing input, comparisons of character special files need not compare all of the data in those files.

For files which are not text files, line numbers simply reflect the presence of a <newline>, without any implication that the file is organized into lines.

Since the behavior of **−s** differs between implementations as to whether error messages are written, the only way to ensure consistent behavior of *cmp* when **−s** is used is to redirect standard error to **/dev/null**.

If error messages are wanted, instead of using **−s** standard output should be redirected to **/dev/null**, and anything written to standard error should be discarded if the exit status is 1. For example:

```
silent_cmp() {
    # compare files with no output except error messages
    message=$(cmp "$@" 2>&1 >/dev/null)
    status=$?
    case $status in
    (0|1) ;;
    (*) printf '%s\n' "$message" ;;
    esac
    return $status
}
```

EXAMPLES

None.

RATIONALE

The global language in Section 1.4 (on page 2462) indicates that using two mutually-exclusive options together produces unspecified results. Some System V implementations consider the option usage:

```
cmp -l -s ...
```

to be an error. They also treat:

```
cmp -s -l ...
```

as if no options were specified. Both of these behaviors are considered bugs, but are allowed.

The word **char** in the standard output format comes from historical usage, even though it is actually a byte number. When *cmp* is supported in other locales, implementations are encouraged to use the word *byte* or its equivalent in another language. Users should not interpret this difference to indicate that the functionality of the utility changed between locales.

Some implementations report on the number of lines in the identical-but-shorter file case. This is allowed by the inclusion of the <*additional info*> fields in the output format. The restriction on having a leading <blank> and no <newline> characters is to make parsing for the filename easier. It is recognized that some filenames containing white-space characters make parsing difficult anyway, but the restriction does aid programs used on systems where the names are predominantly well behaved.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

Future versions of this standard may require that diagnostic messages are written to standard error when the **–s** option is specified.

SEE ALSO

comm, *diff*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 7

SD5-XCU-ERN-96 is applied, updating the STDERR section.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0075 [478] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

comm — select or reject lines common to two files

SYNOPSIS

```
comm [-123] file1 file2
```

DESCRIPTION

The *comm* utility shall read *file1* and *file2*, which should be ordered in the current collating sequence, and produce three text columns as output: lines only in *file1*, lines only in *file2*, and lines in both files.

If the lines in both files are not ordered according to the collating sequence of the current locale, the results are unspecified.

If the collating sequence of the current locale does not have a total ordering of all characters (see XBD Section 7.3.2, on page 139) and any lines from the input files collate equally but are not identical, *comm* shall treat them as different lines and shall expect them to be ordered according to a further byte-by-byte comparison using the collating sequence for the POSIX locale; if they are not ordered in this way, the output of *comm* can identify such lines as being both unique to *file1* and unique to *file2* instead of being in both files.

OPTIONS

The *comm* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−1 Suppress the output column of lines unique to *file1*.

−2 Suppress the output column of lines unique to *file2*.

−3 Suppress the output column of lines duplicated in *file1* and *file2*.

OPERANDS

The following operands shall be supported:

file1 A pathname of the first file to be compared. If *file1* is `'-'`, the standard input shall be used.

file2 A pathname of the second file to be compared. If *file2* is `'-'`, the standard input shall be used.

If both *file1* and *file2* refer to standard input or to the same FIFO special, block special, or character special file, the results are undefined.

STDIN

The standard input shall be used only if one of the *file1* or *file2* operands refers to standard input. See the INPUT FILES section.

INPUT FILES

The input files shall be text files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *comm*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the collating sequence *comm* expects to have been used when the input files were sorted.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS
Default.

STDOUT
The *comm* utility shall produce output depending on the options selected. If the **–1**, **–2**, and **–3** options are all selected, *comm* shall write nothing to standard output.

If the **–1** option is not selected, lines contained only in *file1* shall be written using the format:

`"%s\n", <line in file1>`

If the **–2** option is not selected, lines contained only in *file2* are written using the format:

`"%s%s\n", <lead>, <line in file2>`

where the string **<***lead***>** is as follows:

<tab**>** The **–1** option is not selected.

null string The **–1** option is selected.

If the **–3** option is not selected, lines contained in both files shall be written using the format:

`"%s%s\n", <lead>, <line in both>`

where the string **<***lead***>** is as follows:

<tab**><**tab**>** Neither the **–1** nor the **–2** option is selected.

<tab**>** Exactly one of the **–1** and **–2** options is selected.

null string Both the **–1** and **–2** options are selected.

If the input files were ordered according to the collating sequence of the current locale, the lines written shall be in the collating sequence of the current locale. If the input files contained any lines that collated equally but were not identical and within each file those lines were ordered according to a further byte-by-byte comparison using the collating sequence for the POSIX locale, then lines written that collate equally but are not identical shall be ordered according to a further byte-by-byte comparison using the collating sequence for the POSIX locale.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All input files were successfully output as specified.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

If the input files are not properly presorted, the output of *comm* might not be useful.

When using *comm* to process pathnames, it is recommended that LC_ALL, or at least LC_CTYPE and LC_COLLATE, are set to POSIX or C in the environment, since pathnames can contain byte sequences that do not form valid characters in some locales, in which case the utility's behavior would be undefined. In the POSIX locale each byte is a valid single-byte character, and therefore this problem is avoided.

If the collating sequence of the current locale does not have a total ordering of all characters, since *comm* treats lines as being the same only if they are identical, some lines can be misleadingly identified as being both unique to *file1* and unique to *file2* if lines that collate equally but are not identical are not ordered in the way that *comm* expects. If the input does not come from utilities (such as *ls* and *sort*) which provide this ordering, the problem can be avoided by pre-sorting the input files using *sort*.

EXAMPLES

If a file named **xcu** contains a sorted list of the utilities in this volume of POSIX.1-2024, a file named **xpg3** contains a sorted list of the utilities specified in the X/Open Portability Guide, Issue 3, and a file named **svid89** contains a sorted list of the utilities in the System V Interface Definition Third Edition:

```
comm -23 xcu xpg3 | comm -23 - svid89
```

would print a list of utilities in this volume of POSIX.1-2024 not specified by either of the other documents:

```
comm -12 xcu xpg3 | comm -12 - svid89
```

would print a list of utilities specified by all three documents, and:

```
comm -12 xpg3 svid89 | comm -23 - xcu
```

would print a list of utilities specified by both XPG3 and the SVID, but not specified in this volume of POSIX.1-2024.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cmp, *diff*, *sort*, *uniq*

XBD Section 7.3.2 (on page 139), Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0076 [963], XCU/TC2-2008/0077 [663], and XCU/TC2-2008/0078 [963] are applied.

Issue 8

Austin Group Defect 1070 is applied, changing the requirements when any lines from the input files collate equally but are not identical.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

command — execute a simple command

SYNOPSIS

```
command [-p] command_name [argument...]
command [-p][-v|-V] command_name
```

DESCRIPTION

The *command* utility shall cause the shell to treat the arguments as a simple command, suppressing the shell function lookup that is described in Section 2.9.1.4 (on page 2502), item 1c.

If the *command_name* is the same as the name of one of the special built-in utilities, the special properties in the enumerated list at the beginning of Section 2.15 (on page 2526) shall not occur. In every other respect, if *command_name* is not the name of a function, the effect of *command* (with no options) shall be the same as omitting *command*, except that *command_name* does not appear in the command word position in the *command* command, and consequently is not subject to alias substitution (see Section 2.3.1, on page 2477) nor recognized as a reserved word (see Section 2.4, on page 2478).

When the **–v** or **–V** option is used, the *command* utility shall provide information concerning how a command name is interpreted by the shell.

The *command* utility shall be treated as a declaration utility if the first argument passed to the utility is recognized as a declaration utility. In this case, subsequent words of the form *name=word* shall be expanded in an assignment context. See Section 2.9.1.1 (on page 2500).

OPTIONS

The *command* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–p Perform the command search using a default value for *PATH* that is guaranteed to find all of the standard utilities.

–v Write a string to standard output that indicates the pathname or command that will be used by the shell, in the current shell execution environment (see Section 2.13, on page 2522), to invoke *command_name*, but do not invoke *command_name*.

- Executable utilities, regular built-in utilities, *command_name*s including a <slash> character, and any implementation-provided functions that are found using the *PATH* variable (as described in Section 2.9.1.4, on page 2502), shall be written as absolute pathnames.
- Shell functions, special built-in utilities, regular built-in utilities not associated with a *PATH* search, and shell reserved words shall be written as just their names.
- An alias shall be written as a command line that represents its alias definition.
- Otherwise, no output shall be written and the exit status shall reflect that the name was not found.

–V Write a string to standard output that indicates how the name given in the *command_name* operand will be interpreted by the shell, in the current shell execution environment (see Section 2.13, on page 2522), but do not invoke *command_name*. Although the format of this string is unspecified, it shall indicate in which of the following categories *command_name* falls and shall include the information stated:

- Executable utilities, regular built-in utilities, and any implementation-provided functions that are found using the *PATH* variable (as described in Section 2.9.1.4, on page 2502), shall be identified as such and include the absolute pathname in the string.
- Other shell functions shall be identified as functions.
- Aliases shall be identified as aliases and their definitions included in the string.
- Special built-in utilities shall be identified as special built-in utilities.
- Regular built-in utilities not associated with a *PATH* search shall be identified as regular built-in utilities. (The term ``regular'' need not be used.)
- Shell reserved words shall be identified as reserved words.

OPERANDS

The following operands shall be supported:

argument One of the strings treated as an argument to *command_name*.

command_name
The name of a utility or a special built-in utility.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *command*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

PATH Determine the search path used during the command search described in Section 2.9.1.4 (on page 2502), except as described under the **–p** option.

ASYNCHRONOUS EVENTS

Default.

90161 **STDOUT**

90162 When the **–v** option is specified, standard output shall be formatted as:

90163 `"%s\n", <pathname or command>`

90164 When the **–V** option is specified, standard output shall be formatted as:

90165 `"%s\n", <unspecified>`

90166 **STDERR**

90167 The standard error shall be used only for diagnostic messages.

90168 **OUTPUT FILES**

90169 None.

90170 **EXTENDED DESCRIPTION**

90171 None.

90172 **EXIT STATUS**

90173 When the **–v** or **–V** options are specified, the following exit values shall be returned:

90174 0 Successful completion.

90175 **>**0 The *command_name* could not be found or an error occurred.

90176 Otherwise, the following exit values shall be returned:

90177 126 The utility specified by *command_name* was found but could not be invoked.

90178 127 An error occurred in the *command* utility or the utility specified by *command_name* could not
90179 be found.

90180 Otherwise, the exit status of *command* shall be that of the simple command specified by the
90181 arguments to *command*.

90182 **CONSEQUENCES OF ERRORS**

90183 Default.

90184 **APPLICATION USAGE**

90185 This utility is required to be intrinsic. See Section 1.7 (on page 2470) for details.

90186 The order for command search allows functions to override regular built-ins and path searches.
90187 This utility is necessary to allow functions that have the same name as a utility to call the utility
90188 (instead of a recursive call to the function).

90189 The system default path is available using *getconf*; however, since *getconf* may need to have the
90190 *PATH* set up before it can be called itself, the following can be used:

90191 `command –p getconf PATH`

90192 There are some advantages to suppressing the special characteristics of special built-ins on
90193 occasion. For example:

90194 `command exec > unwritable-file`

90195 does not cause a non-interactive script to abort, so that the output status can be checked by the
90196 script.

90197 The *command*, *env*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit code 127
90198 if a utility to be invoked cannot be found, so that applications can distinguish ``failure to find a
90199 utility'' from ``invoked utility exited with an error indication''. However, the *command* and *nohup*
90200 utilities also use exit code 127 when an error occurs in those utilities, which means exit code 127
90201 is not universally a ``not found'' indicator. The value 127 was chosen because it is not commonly

used for other meanings; most utilities use small values for ``normal error conditions'' and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for any other reason.

Since the **–v** and **–V** options of *command* produce output in relation to the current shell execution environment, *command* is generally provided as a shell regular built-in. If it is called in a subshell or separate utility execution environment, such as one of the following:

```
(PATH=foo command -v)
 nohup command -v
```

it does not necessarily produce correct results. For example, when called with *nohup* or an *exec* function, in a separate utility execution environment, most implementations are not able to identify aliases, functions, or special built-ins.

Two types of regular built-ins could be encountered on a system and these are described separately by *command*. The description of command search in Section 2.9.1.4 (on page 2502) allows for a standard utility to be implemented as a regular built-in as long as it is found in the appropriate place in a *PATH* search. So, for example, *command* **–v** *true* might yield **/bin/true** or some similar pathname. Other implementation-defined utilities that are not defined by this volume of POSIX.1-2024 might exist only as built-ins and have no pathname associated with them. These produce output identified as (regular) built-ins. Applications encountering these are not able to count on *exec*ing them, using them with *nohup*, overriding them with a different *PATH*, and so on.

The *command* utility takes on the expansion behavior of the command that it is wrapping. Therefore, in

```
command command export a=~
```

command is recognized as a declaration utility, and the command sets the variable *a* to the value of `$HOME` because it performs tilde-expansion of an assignment context; while

```
command echo a=~
```

outputs the literal string `"a=~"` because regular expansion can only perform tilde-expansion at the beginning of the word. However, the shell need only perform lexical analysis of the next argument when deciding if command should be treated as a declaration utility; therefore, with:

```
var=export; command $var a=~
```

and

```
command -- export a=~
```

it is unspecified whether the word `a=~` is handled in an assignment context or as a regular expansion.

EXAMPLES

1. Make a version of *cd* that always prints out the new working directory exactly once:

```
cd() {
    command cd "$@" >/dev/null
    pwd
}
```


2. Start off a ``secure shell script'' in which the script avoids being spoofed by its parent:

```
IFS='
'
#    The preceding value should be <space><tab><newline>.
#    Set IFS to its default value.

\unalias -a
#    Unset all possible aliases.
#    Note that unalias is escaped to prevent an alias
#    being used for unalias.

unset -f command
#    Ensure command is not a user function.

PATH="$(command -p getconf PATH):$PATH"
#    Put on a reliable PATH prefix.

#    ...
```

At this point, given correct permissions on the directories called by *PATH*, the script has the ability to ensure that any utility it calls is the intended one. It is being very cautious because it assumes that implementation extensions may be present that would allow user functions to exist when it is invoked; this capability is not specified by this volume of POSIX.1-2024, but it is not prohibited as an extension. For example, the *ENV* variable precedes the invocation of the script with a user start-up script. Such a script could define functions to spoof the application.

RATIONALE

Since *command* is a regular built-in utility it is always found prior to the *PATH* search.

There is nothing in the description of *command* that implies the command line is parsed any differently from that of any other simple command. For example:

```
command a | b ; c
```

is not parsed in any special way that causes `'|'` or `';'` to be treated other than a pipe operator or <semicolon> or that prevents function lookup on **b** or **c**. However, some implementations extend the shell's assignment syntax, for example to allow an array to be populated with a single assignment, and in order for such an extension to be usable in assignments specified as arguments to *export* and *readonly* these shells have those utility names as separate tokens in their grammar. When *command* is used to execute these utilities it also needs to be a separate token in the grammar so that the same extended assignment syntax can still be recognized in this case. This standard only permits an extension of this nature when the input to the shell would contain a syntax error according to the standard grammar, and therefore it cannot affect how `'|'` and `';'` are parsed in the example above. Note that although *command* can be a separate token in the shell's grammar, it cannot be a reserved word since *command* is a candidate for alias substitution whereas reserved words are not (see Section 2.3.1).

The *command* utility is somewhat similar to the Eighth Edition shell *builtin* command, but since *command* also goes to the file system to search for utilities, the name *builtin* would not be intuitive.

The *command* utility is most likely to be provided as a regular built-in. It is not listed as a special built-in for the following reasons:

- The removal of exportable functions made the special precedence of a special built-in unnecessary.
- A special built-in has special properties (see Section 2.15, on page 2526) that were inappropriate for invoking other utilities. For example, two commands such as:

  ```
  date > unwritable-file
  command date > unwritable-file
  ```

  would have entirely different results; in a non-interactive script, the former would continue to execute the next command, the latter would abort. Introducing this semantic difference along with suppressing functions was seen to be non-intuitive.

The **−p** option is present because it is useful to be able to ensure a safe path search that finds all the standard utilities. This search might not be identical to the one that occurs through one of the *exec* functions (as defined in the System Interfaces volume of POSIX.1-2024) when *PATH* is unset. At the very least, this feature is required to allow the script to access the correct version of *getconf* so that the value of the default path can be accurately retrieved.

The *command* **−v** and **−V** options were added to satisfy requirements from users that are currently accomplished by three different historical utilities: *type* in the System V shell, *whence* in the KornShell, and *which* in the C shell. Since there is no historical agreement on how and what to accomplish here, the POSIX *command* utility was enhanced and the historical utilities were left unmodified. The C shell *which* merely conducts a path search. The KornShell *whence* is more elaborate—in addition to the categories required by POSIX, it also reports on tracked aliases, exported aliases, and undefined functions.

The output format of **−V** was left mostly unspecified because human users are its only audience. Applications should not be written to care about this information; they can use the output of **−v** to differentiate between various types of commands, but the additional information that may be emitted by the more verbose **−V** is not needed and should not be arbitrarily constrained in its verbosity or localization for application parsing reasons.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

Section 2.9.1.4 (on page 2502), Section 2.9.1.1 (on page 2500), Section 2.13 (on page 2522), Section 2.15 (on page 2526), *sh*, *type*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *exec*

CHANGE HISTORY

First released in Issue 4.

Issue 7

Austin Group Interpretation 1003.1-2001 #196 is applied, changing the SYNOPSIS to allow **−p** to be used with **−v** (or **−V**).

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The *command* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

The APPLICATION USAGE and EXAMPLES are revised to replace the non-standard *getconf*_CS_PATH with *getconf PATH*.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defects 351 and 1393 are applied, requiring *command* to be a declaration utility if the first argument passed to the utility is recognized as a declaration utility.

Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that this utility is required to be intrinsic.

Austin Group Defect 1117 is applied, changing ``implementation-defined'' to ``implementation-provided''.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1161 is applied, changing ``Utilities'' to ``Executable utilities'' in the descriptions of the **−v** and **−V** options.

Austin Group Defect 1431 is applied, changing ``item 1b'' to ``item 1c''.

Austin Group Defect 1586 is applied, adding the *timeout* utility.

Austin Group Defect 1594 is applied, changing the APPLICATION USAGE section.

Austin Group Defect 1637 is applied, clarifying that (when no options are specified) *command_name* is not subject to alias substitution nor recognized as a reserved word.

NAME

compress, uncompress, zcat — compress and decompress data

SYNOPSIS

XSI

```
compress [-fv] [-b value] [-g | -m algo] [file...]
compress -c [-fv] [-b value] [-g | -m algo] [file]
compress -d [-cfv] [file...]
uncompress [-cfv] [file...]
zcat [file...]
```

DESCRIPTION

The *compress* utility, when the **−d** option is not specified, shall apply the compression algorithm identified by the **−g** option or the **−m** *algo* option to the named files to attempt to reduce their size without loss of information. The compress utility with the **−d** option shall apply the appropriate decompression algorithm to the named files to restore the data to their original state.

The *uncompress* utility shall be equivalent to *compress* **−d**. The *zcat* utility shall be equivalent to *compress* **−c** **−d**. If multiple *file* operands are specified, the decompressed data from each input file shall be concatenated to standard output.

When compressing data, unless the **−c** option is specified, after an input file other than standard input has been compressed, the compressed data from the input file shall be stored in a file with the same pathname as the input file but with an added suffix. The added suffix shall be the suffix associated with the algorithm (see the algorithms in Table 3-7, on page 2737). If appending the suffix would make the size of the last component of the output file's pathname exceed {NAME_MAX} bytes, the command shall fail. If appending the suffix would make the size of the pathname exceed {PATH_MAX} bytes, the command may fail.

When decompressing data, unless the **−c** option is specified, after an input file other than standard input has been decompressed, the decompressed data from the input file shall be stored in a file with the same pathname as the input file but with the suffix associated with the

OB algorithm removed. If *file* has no suffix associated with a known compression algorithm or *file* does not exist and does not have a **.Z** suffix, *file* shall be used as the name of the output file, and the default suffix **.Z** shall be appended to *file* to form the input pathname. The behavior is unspecified if the input pathname ends with a suffix other than the suffix associated with the algorithm used to compress the data. When the **−c** option is specified, *file* can have any suffix, or no suffix, and the utility shall use *file* as the input file and examine the file's contents to determine which algorithm to use to decompress the data (it is not an error if *file* does not have a suffix that matches the suffix associated with the compression algorithm).

When compressing or decompressing a file other than standard input and the **−c** option is not specified, if the invoking process has sufficient privilege, the ownership, modes, access time, and modification time of the output file shall match the ownership, modes, access time, and modification time of the input file. After the output file has been successfully created, the input file shall be removed if the invoking process has sufficient privileges. If the invoking process does not have sufficient privileges to remove the input file (for example, if the directory has the S_ISVTX bit set) the behavior depends on whether the **−f** option is specified: if **−f** is not specified, the output file shall be removed, a diagnostic message shall be written and the utility shall continue processing other files but the final exit status shall be non-zero; if **−f** is specified, the output file shall not be removed and it is unspecified whether the inability to remove the input file is treated as an error. If it is not treated as an error, a warning message may be written

to standard error

If no *file* operands are specified, standard input shall be compressed or decompressed to standard output.

OB If an input file that is to be removed after processing has multiple hard links, the *compress* and *uncompress* utilities may write a diagnostic message to standard error and do nothing with the file; this behavior may depend on whether the **–f** option is specified. If a diagnostic message is written, the final exit status shall be non-zero.

OPTIONS

The *compress*, *uncompress*, and *zcat* utilities shall conform to XBD Section 12.2 (on page 215), except that Guideline 1 does not apply to *uncompress* since the utility name has ten letters.

The following options shall be supported:

–b *value* If the compression algorithm is LZW, *value* specifies the maximum number of bits to use in a code. For a conforming application, the *value* argument shall be:

```
9 <= value <= 16
```

The implementation may allow values of greater than 16. The default shall be 14, 15, or 16.

If the compression algorithm is DEFLATE, *value* specifies the compression level. For a conforming application, the *value* argument shall be:

```
1 <= value <= 9
```

The default shall be 6.

For other algorithms, value specifies implementation-defined tuning.

–c Write to standard output; the input files shall not be changed, and no output files shall be created.

–d Decompress files. When invoked with the **–d** option, the *compress* utility shall restore previously compressed files to their original state.

–f Force compression or decompression of file, even if it does not (for compression) actually reduce the size of the file, or if the corresponding output file already exists. If the **–f** option is not given and the standard input is a terminal, the user shall be prompted as to whether an existing output file should be overwritten. If the response is affirmative, the existing file shall be overwritten. If the standard input is not a terminal and **–f** is not given, *compress* or *uncompress* shall write a diagnostic message to standard error, the existing file shall not be overwritten, and the utility shall exit with a status greater than zero. If the **–f** option is specified and an input file other than standard input has multiple hard links, it is implementation-defined whether the input file is unlinked after the corresponding output file is successfully written, or if processing of that file is skipped and a diagnostic message is written to standard error.

–g Equivalent to **–m** *gzip*.

–m *algo* Use the algorithm defined by *algo* to compress the files. The following algorithms shall be supported:

Table 3-7 Compression algorithms, –**m** option-argument values, and suffixes

Algorithm	*algo*	Filename Suffix
Adaptive LZW	**lzw**	**.Z**
RFC1951 DEFLATE	**deflate**	**.gz**
Synonym for DEFLATE	**gzip**	**.gz**

Other implementation-defined algorithms may be supported.

If neither of the –**m** *algo* and –**g** options is specified, **lzw** shall be used as a default *algo* value. Specifying more than one of the mutually exclusive –**g** and –**m** *algo* options, or multiple –**m** *algo* options, shall not be considered an error. The last option specified shall determine the behavior of the utility.

On systems not supporting the selected algorithm, the input files shall not be changed and an exit status greater than two shall be returned.

Note: The Lempel-Ziv compression algorithm is described in the now-expired US Patent 4464650, which was issued to William Eastman, Abraham Lempel, Jacob Ziv, and Martin Cohn on August 7th, 1984 and assigned to Sperry Corporation.

The Lempel-Ziv-Welch compression algorithm is described in the now-expired US Patent 4558302, which was issued to Terry A. Welch on December 10th, 1985 and assigned to Sperry Corporation.

–**v** For *compress*, write the percentage reduction of each file to standard error. For *uncompress*, write messages to standard error concerning the expansion of each file.

OPERANDS

The following operand shall be supported:

file A pathname of a file to be compressed or decompressed. If a *file* is `'-'`, the utility shall read from standard input at that point in the sequence and write to standard output. If more than one *file* operand is `'-'`, the behavior is unspecified.

STDIN

The standard input shall be used only if no *file* operands are specified, or if a *file* operand is `'-'`.

INPUT FILES

If *file* operands are specified, the corresponding input files contain the data to be compressed or decompressed.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *compress*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements used in the extended regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES* category.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments), the behavior of character classes used in the extended regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES* category.

LC_MESSAGES
Determine the locale used to process affirmative responses, and the locale used to affect the format and contents of diagnostic messages, prompts, and the output from the −**v** option written to standard error.

NLSPATH Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

For the *compress* and *uncompress* utilities, the standard output shall be used if no *file* operands are specified, if a *file* operand is `'-'`, or if the −**c** option is specified. Otherwise, the standard output shall not be used.

The *zcat* utility shall write the decompressed data to the standard output.

STDERR

The standard error shall be used only for diagnostic and prompt messages, the optional warning message described in DESCRIPTION, and the output from −**v**.

OUTPUT FILES

When decompressing input files other than standard input, the corresponding output files shall contain the decompressed input data. When compressing input files other than standard input, the corresponding output files shall contain the compressed input data. If the selected *algo* is **deflate** or **gzip**, the compressed output shall be in the GZIP format described in RFC 1952. For other algorithms, the compressed output file format is implementation-defined and interchange of such files between implementations (including access via unspecified file sharing mechanisms) is not required by POSIX.1-2024.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned for *compress*:

0 Successful completion.

1 An error occurred.

2 One or more files were not compressed because they would have increased in size (and the −**f** option was not specified).

>2 An error occurred.

The following exit values shall be returned for *uncompress* and *zcat*:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

If an error occurs while compressing or decompressing an input file other than standard input, the input file shall remain unmodified.

APPLICATION USAGE

The amount of compression obtained depends on the size of the input, the number of bits per code, and the distribution of common substrings. Typically, text such as source code or English is reduced by 50-60%. Compression is generally much better than that achieved by Huffman coding or adaptive Huffman coding (*compact*), and takes less time to compute.

Although *compress* strictly follows the default actions upon receipt of a signal or when an error occurs, some unexpected results may occur. In some implementations it is likely that a partially compressed file is left in place, alongside its uncompressed input file. Since the general operation of *compress* is to delete the uncompressed file only after the **.Z** file has been successfully filled, an application should always carefully check the exit status of *compress* before arbitrarily deleting files that have like-named neighbors with **.Z** suffixes.

In addition to trying *file* and *file***.Z** when looking for a file to decompress, some implementations of *uncompress* and *zcat* also try suffixes for other known compression algorithms if neither *file* nor *file***.Z** is found. This version of the standard allows, but does not require this behavior. Portable applications should always specify the full pathname (including the suffix) of files to be decompressed.

EXAMPLES

None.

RATIONALE

Earlier versions of this standard limited the number of bits used by conforming applications for the **lzw** algorithm to 14 due to address space limitations on 16-bit architectures. Using 15 or 16 is a much more common default when using current hardware.

Earlier versions of this standard only supported LZW compression. The standard developers noted that existing implementations added other compression utilities, such as *gzip*, and found it desirable to support this widespread usage. Some implementations had extended the *compress* utility to support such other schemes. The standard developers generalized this practice by the addition of the **−m** option, even though this was not previous practice.

The *uncompress* **−d** option is added to match undocumented existing practice of tested implementations.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

When decompressing a file, the requirement to add **.Z** to a *file* operand if the given pathname does not include a suffix associated with a known compression algorithm or if *file* does not exist and does not already have a **.Z** extension is an obsolescent feature and may be removed in a future version.

SEE ALSO

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 6

The normative text is reworded to avoid use of the term ``must'' for application requirements.

An error case is added for systems not supporting adaptive Lempel-Ziv coding.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Austin Group Interpretation 1003.1-2001 #125 is applied, revising the ENVIRONMENT VARIABLES section.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1041 is applied, combining the *compress*, *uncompress* and *zcat* pages into one and extensively modifying most sections.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

cp — copy files

SYNOPSIS

```
cp [-Pfip] source_file target_file
cp [-Pfip] source_file... target
cp -R [-H|-L|-P] [-fip] source_file... target
```

DESCRIPTION

The first synopsis form is denoted by two operands, neither of which are existing files of type directory. The *cp* utility shall copy the contents of *source_file* (or, if *source_file* is a file of type symbolic link, the contents of the file referenced by *source_file*) to the destination path named by *target_file.*

The second synopsis form is denoted by two or more operands where the **–R** option is not specified and the first synopsis form is not applicable. It shall be an error if any *source_file* is a file of type directory, if *target* does not exist, or if *target* does not name a directory. The *cp* utility shall copy the contents of each *source_file* (or, if *source_file* is a file of type symbolic link, the contents of the file referenced by *source_file*) to the destination path named by the concatenation of *target*, a single <slash> character if *target* did not end in a <slash>, and the last component of *source_file*.

The third synopsis form is denoted by two or more operands where the **–R** option is specified. The *cp* utility shall copy each file in the file hierarchy rooted in each *source_file* to a destination path named as follows:

- If *target* exists and names an existing directory, the name of the corresponding destination path for each file in the file hierarchy shall be the concatenation of *target*, a single <slash> character if *target* did not end in a <slash>, and the pathname of the file relative to the directory containing *source_file*.
- If *target* does not exist and two operands are specified, the name of the corresponding destination path for *source_file* shall be *target*; the name of the corresponding destination path for all other files in the file hierarchy shall be the concatenation of *target*, a <slash> character, and the pathname of the file relative to *source_file*.

It shall be an error if *target* does not exist and more than two operands are specified, or if *target* exists and does not name a directory.

In the following description, the term *dest_file* refers to the file named by the destination path. The term *source_file* refers to the file that is being copied, whether specified as an operand or a file in a file hierarchy rooted in a *source_file* operand. If *source_file* is a file of type symbolic link:

- If the **–R** option was not specified, *cp* shall take actions based on the type and contents of the file referenced by the symbolic link, and not by the symbolic link itself, unless the **–P** option was specified.
- If the **–R** option was specified:
  - — If none of the options **–H**, **–L**, nor **–P** were specified, it is unspecified which of **–H**, **–L**, or **–P** will be used as a default.
  - — If the **–H** option was specified, *cp* shall take actions based on the type and contents of the file referenced by any symbolic link specified as a *source_file* operand.
  - — If the **–L** option was specified, *cp* shall take actions based on the type and contents of the file referenced by any symbolic link specified as a *source_file* operand or any symbolic links encountered during traversal of a file hierarchy.

— If the **−P** option was specified, *cp* shall copy any symbolic link specified as a *source_file* operand and any symbolic links encountered during traversal of a file hierarchy, and shall not follow any symbolic links.

For each *source_file*, the following steps shall be taken:

1. If *source_file* references the same file as *dest_file*, *cp* may write a diagnostic message to standard error; it shall do nothing more with *source_file* and shall go on to any remaining files.
2. If *source_file* is of type directory, the following steps shall be taken:
   a. If the **−R** option was not specified, *cp* shall write a diagnostic message to standard error, do nothing more with *source_file*, and go on to any remaining files.
   b. If *source_file* was not specified as an operand and *source_file* is dot or dot-dot, *cp* shall do nothing more with *source_file* and go on to any remaining files.
   c. If *dest_file* exists and it is a file type not specified by the System Interfaces volume of POSIX.1-2024, the behavior is implementation-defined.
   d. If *dest_file* exists and it is not of type directory, *cp* shall write a diagnostic message to standard error, do nothing more with *source_file* or any files below *source_file* in the file hierarchy, and go on to any remaining files.
   e. If the directory *dest_file* does not exist, it shall be created with file permission bits set to the same value as those of *source_file*, modified by the file creation mask of the user if the **−p** option was not specified, and then bitwise-inclusively OR'ed with S_IRWXU. If *dest_file* cannot be created, *cp* shall write a diagnostic message to standard error, do nothing more with *source_file*, and go on to any remaining files. It is unspecified if *cp* attempts to copy files in the file hierarchy rooted in *source_file*.
   f. The files in the directory *source_file* shall be copied to the directory *dest_file*, taking the four steps (1 to 4) listed here with the files as *source_file*s.
   g. If *dest_file* was created, its file permission bits shall be changed (if necessary) to be the same as those of *source_file*, modified by the file creation mask of the user if the **−p** option was not specified.
   h. The *cp* utility shall do nothing more with *source_file* and go on to any remaining files.
3. If *source_file* is of type regular file, the following steps shall be taken:
   a. The behavior is unspecified if *dest_file* exists and was written by a previous step. Otherwise, if *dest_file* exists, the following steps shall be taken:
      i. If the **−i** option is in effect, the *cp* utility shall write a prompt to the standard error and read a line from the standard input. If the response is not affirmative, *cp* shall do nothing more with *source_file* and go on to any remaining files.
      ii. A file descriptor for *dest_file* shall be obtained by performing actions equivalent to the *open*( ) function defined in the System Interfaces volume of POSIX.1-2024 called using *dest_file* as the *path* argument, and the bitwise-inclusive OR of O_WRONLY and O_TRUNC as the *oflag* argument.
      iii. If the attempt to obtain a file descriptor fails and the **−f** option is in effect, *cp* shall attempt to remove the file by performing actions equivalent to the *unlink*( ) function defined in the System Interfaces volume of POSIX.1-2024

called using *dest_file* as the *path* argument. If this attempt succeeds, *cp* shall continue with step 3b.

b. If *dest_file* does not exist, a file descriptor shall be obtained by performing actions equivalent to the *open*() function defined in the System Interfaces volume of POSIX.1-2024 called using *dest_file* as the *path* argument, and the bitwise-inclusive OR of O_WRONLY and O_CREAT as the *oflag* argument. The file permission bits of *source_file* shall be the *mode* argument.

c. If the attempt to obtain a file descriptor fails, *cp* shall write a diagnostic message to standard error, do nothing more with *source_file*, and go on to any remaining files.

d. The contents of *source_file* shall be written to the file descriptor. Any write errors shall cause *cp* to write a diagnostic message to standard error and continue to step 3e.

e. The file descriptor shall be closed.

f. The *cp* utility shall do nothing more with *source_file*. If a write error occurred in step 3d, it is unspecified if *cp* continues with any remaining files. If no write error occurred in step 3d, *cp* shall go on to any remaining files.

4. Otherwise, the **–R** option was specified, and the following steps shall be taken:

a. The *dest_file* shall be created with the same file type as *source_file*.

b. If *source_file* is a file of type FIFO, the file permission bits shall be the same as those of *source_file,* modified by the file creation mask of the user if the **–p** option was not specified. Otherwise, the permissions, owner ID, and group ID of *dest_file* are implementation-defined.

If this creation fails for any reason, *cp* shall write a diagnostic message to standard error, do nothing more with *source_file*, and go on to any remaining files.

c. If *source_file* is a file of type symbolic link, and the options require the symbolic link itself to be acted upon, the pathname contained in *dest_file* shall be the same as the pathname contained in *source_file*.

If this fails for any reason, *cp* shall write a diagnostic message to standard error, do nothing more with *source_file*, and go on to any remaining files.

If the implementation provides additional or alternate access control mechanisms (see XBD Section 4.7, on page 97), their effect on copies of files is implementation-defined.

OPTIONS

The *cp* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–f If a file descriptor for a destination file cannot be obtained, as described in step 3.a.ii., attempt to unlink the destination file and proceed.

–H Take actions based on the type and contents of the file referenced by any symbolic link specified as a *source_file* operand.

–i Write a prompt to standard error before copying to any existing non-directory destination file. If the response from the standard input is affirmative, the copy shall be attempted; otherwise, it shall not.

90706 **−L** Take actions based on the type and contents of the file referenced by any symbolic
90707 link specified as a *source_file* operand or any symbolic links encountered during
90708 traversal of a file hierarchy.

90709 **−P** Take actions on any symbolic link specified as a *source_file* operand or any
90710 symbolic link encountered during traversal of a file hierarchy.

90711 **−p** Duplicate the following characteristics of each source file in the corresponding
90712 destination file:

90713 1. The time of last data modification and time of last access. If this duplication
90714 fails for any reason, *cp* shall write a diagnostic message to standard error.

90715 2. The user ID and group ID. If this duplication fails for any reason, it is
90716 unspecified whether *cp* writes a diagnostic message to standard error.

90717 3. The file permission bits and the S_ISUID and S_ISGID bits. Other,
90718 implementation-defined, bits may be duplicated as well. If this duplication
90719 fails for any reason, *cp* shall write a diagnostic message to standard error.

90720 If the user ID or the group ID cannot be duplicated, the file permission bits
90721 S_ISUID and S_ISGID shall be cleared. If these bits are present in the source file but
90722 are not duplicated in the destination file, it is unspecified whether *cp* writes a
90723 diagnostic message to standard error.

90724 The order in which the preceding characteristics are duplicated is unspecified. The
90725 *dest_file* shall not be deleted if these characteristics cannot be preserved.

90726 **−R** Copy file hierarchies.

90727 Specifying more than one of the mutually-exclusive options **−H**, **−L**, and **−P** shall not be
90728 considered an error. The last option specified shall determine the behavior of the utility.

90729 **OPERANDS**

90730 The following operands shall be supported:

90731 *source_file* A pathname of a file to be copied. If a *source_file* operand is `'−'`, it shall refer to a
90732 file named −; implementations shall not treat it as meaning standard input.

90733 *target_file* A pathname of an existing or nonexistent file, used for the output when a single
90734 file is copied. If a *target_file* operand is `'−'`, it shall refer to a file named −;
90735 implementations shall not treat it as meaning standard output.

90736 *target* A pathname of a directory to contain the copied files.

90737 **STDIN**

90738 The standard input shall be used to read an input line in response to each prompt specified in
90739 the STDERR section. Otherwise, the standard input shall not be used.

90740 **INPUT FILES**

90741 The input files specified as operands may be of any file type.

90742 **ENVIRONMENT VARIABLES**

90743 The following environment variables shall affect the execution of *cp*:

90744 *LANG* Provide a default value for the internationalization variables that are unset or null.
90745 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
90746 variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements used in the extended regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES* category.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes used in the extended regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES* category.

LC_MESSAGES
Determine the locale used to process affirmative responses, and the locale used to affect the format and contents of diagnostic messages and prompts written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS
Default.

STDOUT
Not used.

STDERR
A prompt shall be written to standard error under the conditions specified in the DESCRIPTION section. The prompt shall contain the destination pathname, but its format is otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic messages.

OUTPUT FILES
The output files may be of any type.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 All requested files (excluding files where a non-affirmative response was given to a request for confirmation) were successfully copied.

>0 An error occurred.

CONSEQUENCES OF ERRORS
If *cp* is prematurely terminated by a signal or error, files or file hierarchies may be only partially copied and files and directories may have incorrect permissions or access and modification times.

APPLICATION USAGE

The set-user-ID and set-group-ID bits are explicitly cleared when files are created. This is to prevent users from creating programs that are set-user-ID or set-group-ID to them when copying files or to make set-user-ID or set-group-ID files accessible to new groups of users. For example, if a file is set-user-ID and the copy has a different group ID than the source, a new group of users has execute permission to a set-user-ID program than did previously. In particular, this is a problem for superusers copying users' trees.

EXAMPLES

None.

RATIONALE

The **–i** option exists on BSD systems, giving applications and users a way to avoid accidentally removing files when copying. Although the 4.3 BSD version does not prompt if the standard input is not a terminal, the standard developers decided that use of **–i** is a request for interaction, so when the destination path exists, the utility takes instructions from whatever responds on standard input.

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified because implementations may desire more descriptive prompts than those used on historical implementations. Therefore, an application using the **–i** option relies on the system to provide the most suitable dialog directly with the user, based on the behavior specified.

The **–p** option is historical practice on BSD systems, duplicating the time of last data modification and time of last access. This volume of POSIX.1-2024 extends it to preserve the user and group IDs, as well as the file permissions. This requirement has obvious problems in that the directories are almost certainly modified after being copied. This volume of POSIX.1-2024 requires that the modification times be preserved. The statement that the order in which the characteristics are duplicated is unspecified is to permit implementations to provide the maximum amount of security for the user. Implementations should take into account the obvious security issues involved in setting the owner, group, and mode in the wrong order or creating files with an owner, group, or mode different from the final value.

It is unspecified whether *cp* writes diagnostic messages when the user and group IDs cannot be set due to the widespread practice of users using **–p** to duplicate some portion of the file characteristics, indifferent to the duplication of others. Historic implementations only write diagnostic messages on errors other than [EPERM].

Earlier versions of this standard included support for the **–r** option to copy file hierarchies. The **–r** option is historical practice on BSD and BSD-derived systems. This option is no longer specified by POSIX.1-2024 but may be present in some implementations. The **–R** option was added as a close synonym to the **–r** option, selected for consistency with all other options in this volume of POSIX.1-2024 that do recursive directory descent.

The difference between **–R** and the removed **–r** option is in the treatment by *cp* of file types other than regular and directory. It was implementation-defined how the – option treated special files to allow both historical implementations and those that chose to support **–r** with the same abilities as **–R** defined by this volume of POSIX.1-2024. The original **–r** flag, for historic reasons, did not handle special files any differently from regular files, but always read the file and copied its contents. This had obvious problems in the presence of special file types; for example, character devices, FIFOs, and sockets.

When a failure occurs during the copying of a file hierarchy, *cp* is required to attempt to copy files that are on the same level in the hierarchy or above the file where the failure occurred. It is unspecified if *cp* shall attempt to copy files below the file where the failure occurred (which

cannot succeed in any case).

Permissions, owners, and groups of created special file types have been deliberately left as implementation-defined. This is to allow systems to satisfy special requirements (for example, allowing users to create character special devices, but requiring them to be owned by a certain group). In general, it is strongly suggested that the permissions, owner, and group be the same as if the user had run the historical *mknod*, *ln*, or other utility to create the file. It is also probable that additional privileges are required to create block, character, or other implementation-defined special file types.

Additionally, the **−p** option explicitly requires that all set-user-ID and set-group-ID permissions be discarded if any of the owner or group IDs cannot be set. This is to keep users from unintentionally giving away special privilege when copying programs.

When creating regular files, historical versions of *cp* use the mode of the source file as modified by the file mode creation mask. Other choices would have been to use the mode of the source file unmodified by the creation mask or to use the same mode as would be given to a new file created by the user (plus the execution bits of the source file) and then modify it by the file mode creation mask. In the absence of any strong reason to change historic practice, it was in large part retained.

When creating directories, historical versions of *cp* use the mode of the source directory, plus read, write, and search bits for the owner, as modified by the file mode creation mask. This is done so that *cp* can copy trees where the user has read permission, but the owner does not. A side-effect is that if the file creation mask denies the owner permissions, *cp* fails. Also, once the copy is done, historical versions of *cp* set the permissions on the created directory to be the same as the source directory, unmodified by the file creation mask.

This behavior has been modified so that *cp* is always able to create the contents of the directory, regardless of the file creation mask. After the copy is done, the permissions are set to be the same as the source directory, as modified by the file creation mask. This latter change from historical behavior is to prevent users from accidentally creating directories with permissions beyond those they would normally set and for consistency with the behavior of *cp* in creating files.

It is not a requirement that *cp* detect attempts to copy a file to itself; however, implementations are strongly encouraged to do so. Historical implementations have detected the attempt in most cases.

There are two methods of copying subtrees in this volume of POSIX.1-2024. The other method is described as part of the *pax* utility (see *pax*). Both methods are historical practice. The *cp* utility provides a simpler, more intuitive interface, while *pax* offers a finer granularity of control. Each provides additional functionality to the other; in particular, *pax* maintains the hard-link structure of the hierarchy, while *cp* does not. It is the intention of the standard developers that the results be similar (using appropriate option combinations in both utilities). The results are not required to be identical; there seemed insufficient gain to applications to balance the difficulty of implementations having to guarantee that the results would be exactly identical.

The wording allowing *cp* to copy a directory to implementation-defined file types not specified by the System Interfaces volume of POSIX.1-2024 is provided so that implementations supporting symbolic links are not required to prohibit copying directories to symbolic links. Other extensions to the System Interfaces volume of POSIX.1-2024 file types may need to use this loophole as well.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

mv, *find*, *ln*, *pax*

XBD Section 4.7 (on page 97), Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *open*( ), *unlink*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

The –**r** option is marked obsolescent.

The new options –**H**, –**L**, and –**P** are added to align with the IEEE P1003.2b draft standard. These options affect the processing of symbolic links.

IEEE PASC Interpretation 1003.2 #194 is applied, adding a description of the –**P** option.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/18 is applied, correcting an error in the SEE ALSO section.

Issue 7

Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the *LC_MESSAGES* environment variable.

Austin Group Interpretations 1003.1-2001 #092, #164, #165, and #168 are applied.

SD5-XCU-ERN-31 and SD5-XCU-ERN-42 are applied, updating the DESCRIPTION.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

SD5-XCU-ERN-102 is applied, clarifying the –**i** option within the OPTIONS section.

The obsolescent –**r** option is removed.

The –**P** option is added to the SYNOPSIS and to the DESCRIPTION with respect to the –**R** option.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1732 is applied, changing the EXIT STATUS section.

NAME

crontab — schedule periodic background work

SYNOPSIS

```
crontab [file]
```

UP

```
crontab [-e|-l|-r]
```

DESCRIPTION

UP The *crontab* utility shall create, replace, or edit a user's crontab entry; a crontab entry is a list of commands and the times at which they shall be executed. The new crontab entry can be input by specifying *file* or input from standard input if no *file* operand is specified, or by using an editor, if **–e** is specified.

Upon execution of a command from a crontab entry, the implementation shall supply a default environment, defining at least the following environment variables:

HOME A pathname of the user's home directory.

LOGNAME The user's login name.

PATH A string representing a search path guaranteed to find all of the standard utilities.

SHELL A pathname of the command interpreter. When *crontab* is invoked as specified by this volume of POSIX.1-2024, the value shall be a pathname for *sh*.

The values of these variables when *crontab* is invoked as specified by this volume of POSIX.1-2024 shall not affect the default values provided when the scheduled command is run.

If standard output and standard error are not redirected by commands executed from the crontab entry, any generated output or errors shall be mailed, via an implementation-defined method, to the user.

XSI Users shall be permitted to use *crontab* if their names appear in the file **cron.allow** which is located in an implementation-defined directory. If that file does not exist, the file **cron.deny**, which is located in an implementation-defined directory, shall be checked to determine whether the user shall be denied access to *crontab*. If neither file exists, only a process with appropriate privileges shall be allowed to submit a job. If only **cron.deny** exists and is empty, global usage shall be permitted. The **cron.allow** and **cron.deny** files shall consist of one user name per line.

OPTIONS

The *crontab* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

UP **–e** Edit a copy of the invoking user's crontab entry, or create an empty entry to edit if the crontab entry does not exist. When editing is complete, the entry shall be installed as the user's crontab entry.

–l (The letter ell.) List the invoking user's crontab entry.

–r Remove the invoking user's crontab entry.

OPERANDS

The following operand shall be supported:

file The pathname of a file that contains specifications, in the format defined in the INPUT FILES section, for crontab entries.

90948 **STDIN**

90949 See the INPUT FILES section.

90950 **INPUT FILES**

90951 In the POSIX locale, the user or application shall ensure that a crontab entry is a text file
90952 consisting of lines of six fields each. The fields shall be separated by <blank> characters. The
90953 first five fields shall be integer patterns that specify the following:

90954 1. Minute [0,59]
90955 2. Hour [0,23]
90956 3. Day of the month [1,31]
90957 4. Month of the year [1,12]
90958 5. Day of the week ([0,6] with 0=Sunday)

90959 Each of these patterns can be either an <asterisk> (meaning all valid values), an element, or a list
90960 of elements separated by <comma> characters. An element shall be either a number or two
90961 numbers separated by a <hyphen-minus> (meaning an inclusive range). The specification of
90962 days can be made by two fields (day of the month and day of the week). If month, day of month,
90963 and day of week are all <asterisk> characters, every day shall be matched. If either the month or
90964 day of month is specified as an element or list, but the day of week is an <asterisk>, the month
90965 and day of month fields shall specify the days that match. If both month and day of month are
90966 specified as an <asterisk>, but day of week is an element or list, then only the specified days of
90967 the week match. Finally, if either the month or day of month is specified as an element or list,
90968 and the day of week is also specified as an element or list, then any day matching either the
90969 month and day of month, or the day of week, shall be matched.

90970 The sixth field of a line in a crontab entry is a string that shall be executed by *sh* at the specified
90971 times. A <percent-sign> character in this field shall be translated to a <newline>. Any character
90972 preceded by a <backslash> (including the `'%'`) shall cause that character to be treated literally.
90973 Only the first line (up to a `'%'` or end-of-line) of the command field shall be executed by the
90974 command interpreter. The other lines shall be made available to the command as standard input.

90975 Blank lines and those whose first non-<blank> is `'#'` shall be ignored.

90976 XSI The text files **cron.allow** and **cron.deny**, which are located in an implementation-defined
90977 directory, shall contain zero or more user names, one per line, of users who are, respectively,
90978 authorized or denied access to the service underlying the *crontab* utility.

90979 **ENVIRONMENT VARIABLES**

90980 The following environment variables shall affect the execution of *crontab*:

90981 *EDITOR* Determine the editor to be invoked when the **–e** option is specified. The default
90982 editor shall be *vi*.

90983 *LANG* Provide a default value for the internationalization variables that are unset or null.
90984 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
90985 variables used to determine the values of locale categories.)

90986 *LC_ALL* If set to a non-empty string value, override the values of all the other
90987 internationalization variables.

90988 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
90989 characters (for example, single-byte as opposed to multi-byte characters in
90990 arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS
Default.

STDOUT
If the **–l** option is specified, the crontab entry shall be written to the standard output.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS
UP The user's crontab entry is not submitted, removed, edited, or listed.

APPLICATION USAGE
The format of the crontab entry shown here is guaranteed only for the POSIX locale. Other cultures may be supported with substantially different interfaces, although implementations are encouraged to provide comparable levels of functionality.

The default settings of the *HOME*, *LOGNAME*, *PATH*, and *SHELL* variables that are given to the scheduled job are not affected by the settings of those variables when *crontab* is run; as stated, they are defaults. The text about ``invoked as specified by this volume of POSIX.1-2024'' means that the implementation may provide extensions that allow these variables to be affected at runtime, but that the user has to take explicit action in order to access the extension, such as give a new option flag or modify the format of the crontab entry.

A typical user error is to type only *crontab*; this causes the system to wait for the new crontab entry on standard input. If end-of-file is typed (generally <control>-D), the crontab entry is replaced by an empty file. In this case, the user should type the interrupt character, which prevents the crontab entry from being replaced.

EXAMPLES

1. Clean up files named **core** every weekday morning at 3:15 am:

```
15 3 * * 1–5 find "$HOME" –name core –exec rm –f {} + 2>/dev/null
```

2. Mail a birthday greeting:

```
0 12 14 2 * mailx john%Happy Birthday!%Time for lunch.
```

3. As an example of specifying the two types of days:

```
0 0 1,15 * 1
```


would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to `'*'`; for example:

```
0 0 * * 1
```

would run a command only on Mondays.

RATIONALE

All references to a *cron* daemon and to *cron files* have been omitted. Although historical implementations have used this arrangement, there is no reason to limit future implementations.

This description of *crontab* is designed to support only users with normal privileges. The format of the input is based on the System V *crontab*; however, there is no requirement here that the actual system database used by the *cron* daemon (or a similar mechanism) use this format internally. For example, systems derived from BSD are likely to have an additional field appended that indicates the user identity to be used when the job is submitted.

The **–e** option was adopted from the SVID as a user convenience, although it does not exist in all historical implementations.

FUTURE DIRECTIONS

None.

SEE ALSO

at

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

The *crontab* utility (except for the **–e** option) is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced by the *crontab* utility.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The first example is changed to remove the unreliable use of *find* | *xargs*.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0079 [584] is applied.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1141 is applied, changing ``core files'' to ``files named core''.

NAME

csplit — split files based on context

SYNOPSIS

```
csplit [-ks] [-f prefix] [-n number] file arg...
```

DESCRIPTION

The *csplit* utility shall read the file named by the *file* operand, write all or part of that file into other files as directed by the *arg* operands, and write the sizes of the files.

OPTIONS

The *csplit* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−f *prefix* Name the created files *prefix***00**, *prefix***01**, ..., *prefix**n*. The default is **xx00** ... **xx***n*. If the *prefix* argument would create a filename exceeding {NAME_MAX} bytes, an error shall result, *csplit* shall exit with a diagnostic message, and no files shall be created.

−k Leave previously created files intact. By default, *csplit* shall remove created files if an error occurs.

−n *number* Use *number* decimal digits to form filenames for the file pieces. The default shall be 2.

−s Suppress the output of file size messages.

OPERANDS

The following operands shall be supported:

file The pathname of a text file to be split. If *file* is `'-'`, the standard input shall be used.

Each *arg* operand can be one of the following:

*/rexp/***[***offset***]**

A file shall be created using the content of the lines from the current line up to, but not including, the line that results from the evaluation of the regular expression with *offset*, if any, applied. The regular expression *rexp* shall follow the rules for basic regular expressions described in XBD Section 9.3 (on page 181). The application shall use the sequence `"\/"` to specify a <slash> character within the *rexp*. The optional offset shall be a positive or negative integer value representing a number of lines. A positive integer value can be preceded by `'+'`. If the selection of lines from an *offset* expression of this type would create a file with zero lines, or one with greater than the number of lines left in the input file, the results are unspecified. After the section is created, the current line shall be set to the line that results from the evaluation of the regular expression with any offset applied. If the current line is the first line in the file and a regular expression operation has not yet been performed, the pattern match of *rexp* shall be applied from the current line to the end of the file. Otherwise, the pattern match of *rexp* shall be applied from the line following the current line to the end of the file.

*%rexp%***[***offset***]**

Equivalent to */rexp/***[***offset***]**, except that no file shall be created for the selected section of the input file. The application shall use the sequence `"\%"` to specify a <percent-sign> character within the *rexp*.

line_no Create a file from the current line up to (but not including) the line number *line_no*. Lines in the file shall be numbered starting at one. The current line becomes *line_no*.

{*num*} Repeat operand. This operand can follow any of the operands described previously. If it follows a *rexp* type operand, that operand shall be applied *num* more times. If it follows a *line_no* operand, the file shall be split every *line_no* lines, *num* times, from that point.

An error shall be reported if an operand does not reference a line between the current position and the end of the file.

STDIN

See the INPUT FILES section.

INPUT FILES

The input file shall be a text file.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *csplit*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes within regular expressions.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

If the **–k** option is specified, created files shall be retained. Otherwise, the default action occurs.

STDOUT

Unless the **–s** option is used, the standard output shall consist of one line per file created, with a format as follows:

```
"%d\n", <file size in bytes>
```

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

The output files shall contain portions of the original input file; otherwise, unchanged.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

By default, created files shall be removed if an error occurs. When the **–k** option is specified, created files shall not be removed if an error occurs.

APPLICATION USAGE

None.

EXAMPLES

1. This example creates four files, **cobol00** . . . **cobol03**:

   ```
   csplit –f cobol file '/procedure division/' /par5./ /par16./
   ```

   After editing the split files, they can be recombined as follows:

   ```
   cat cobol0[0–3] > file
   ```

   Note that this example overwrites the original file.

2. This example would split the file after the first 99 lines, and every 100 lines thereafter, up to 9 999 lines; this is because lines in the file are numbered from 1 rather than zero, for historical reasons:

   ```
   csplit –k file  100  {99}
   ```

3. Assuming that **prog.c** follows the C-language coding convention of ending routines with a `'}'` at the beginning of the line, this example creates a file containing each separate C routine (up to 21) in **prog.c**:

   ```
   csplit –k prog.c '%main(%'  '/^}/+1' {20}
   ```

RATIONALE

The **–n** option was added to extend the range of filenames that could be handled.

Consideration was given to adding a **–a** flag to use the alphabetic filename generation used by the historical *split* utility, but the functionality added by the **–n** option was deemed to make alphabetic naming unnecessary.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

sed, *split*

XBD Chapter 8 (on page 167), Section 9.3 (on page 181), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The APPLICATION USAGE section is added.

The description of regular expression operands is changed to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

The *csplit* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The SYNOPSIS and OPERANDS sections are revised to use a single *arg* to split a file into two pieces.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

ctags — create a tags file (**DEVELOPMENT**)

SYNOPSIS

CD SD

```
ctags [-a] [-f tagsfile] pathname...
ctags -x pathname...
```

DESCRIPTION

The *ctags* utility shall write a *tagsfile* or an index of objects from C-language source files specified by the *pathname* operands. The *tagsfile* shall list the locators of C-language objects within the source files. A locator consists of a name, pathname, and either a search pattern or a line number that can be used in searching for the object definition. The objects that shall be recognized are specified in the EXTENDED DESCRIPTION section.

OPTIONS

The *ctags* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−a Append to *tagsfile*.

−f *tagsfile* Write the object locator lists into *tagsfile* instead of the default file named **tags** in the current directory.

−x Produce a list of object names, the line number, and filename in which each is defined, as well as the text of that line, and write this to the standard output. A *tagsfile* shall not be created when **−x** is specified.

OPERANDS

The following *pathname* operands are supported:

*file***.c** Files with basenames ending with the **.c** suffix shall be treated as C-language source code. Such files that are not valid input to *c17* produce unspecified results.

*file***.h** Files with basenames ending with the **.h** suffix shall be treated as C-language source code. Such files that are not valid input to *c17* produce unspecified results.

The handling of other files is implementation-defined.

STDIN

See the INPUT FILES section.

INPUT FILES

The input files shall be text files containing C-language source code.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *ctags*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the order in which output is sorted for the **−x** option. The POSIX locale determines the order in which the *tagsfile* is written.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files). If the locale is not compatible with the C locale described by the ISO C standard, the results are unspecified.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The list of object name information produced by the **–x** option shall be written to standard output in the following format:

```
"%s %d %s %s", <object-name>, <line-number>, <filename>, <text>
```

where <*text*> is the text of line <*line-number*> of file <*filename*>.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

When the **–x** option is not specified, the format of the output file shall be:

```
"%s\t%s\t/%s/\n", <identifier>, <filename>, <pattern>
```

where <*pattern*> is a search pattern that could be used by an editor to find the defining instance of <*identifier*> in <*filename*> (where *defining instance* is indicated by the declarations listed in the EXTENDED DESCRIPTION).

An optional <circumflex> (`'^'`) can be added as a prefix to <*pattern*>, and an optional <dollar-sign> can be appended to <*pattern*> to indicate that the pattern is anchored to the beginning (end) of a line of text. Any <slash> or <backslash> characters in <*pattern*> shall be preceded by a <backslash> character. The anchoring <circumflex>, <dollar-sign>, and escaping <backslash> characters shall not be considered part of the search pattern. All other characters in the search pattern shall be considered literal characters.

An alternative format is:

```
"%s\t%s\t?%s?\n", <identifier>, <filename>, <pattern>
```

which is identical to the first format except that <slash> characters in <*pattern*> shall not be preceded by escaping <backslash> characters, and <question-mark> characters in <*pattern*> shall be preceded by <backslash> characters.

A second alternative format is:

```
"%s\t%s\t%d\n", <identifier>, <filename>, <lineno>
```

where <*lineno*> is a decimal line number that could be used by an editor to find <*identifier*> in <*filename*>.

Neither alternative format shall be produced by *ctags* when it is used as described by POSIX.1-2024, but the standard utilities that process tags files shall be able to process those formats as well as the first format.

In any of these formats, the file shall be sorted by identifier, based on the collation sequence in

the POSIX locale.

EXTENDED DESCRIPTION

The *ctags* utility shall attempt to produce an output line for each of the following objects:

- Function definitions
- Type definitions
- Macros with arguments

It may also produce output for any of the following objects:

- Function prototypes
- Structures
- Unions
- Global variable definitions
- Enumeration types
- Macros without arguments
- **#define** statements
- **#line** statements

Any **#if** and **#ifdef** statements shall produce no output. The tag **main** is treated specially in C programs. The tag formed shall be created by prefixing **M** to the name of the file, with the trailing **.c**, and leading pathname components (if any) removed.

It is implementation-defined what other objects (including duplicate identifiers) produce output.

On systems that do not support the C-Language Development Utilities option, if *ctags* is supported it produces unspecified results for C-language source code files. It should write to standard error a message identifying this condition and cause a non-zero exit status to be produced.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The output with –**x** is meant to be a simple index that can be written out as an off-line readable function index. If the input files to *ctags* (such as **.c** files) were not created using the same locale as that in effect when *ctags* –**x** is run, results might not be as expected.

The description of C-language processing says ``attempts to'' because the C language can be greatly confused, especially through the use of **#define**s, and this utility would be of no use if the real C preprocessor were run to identify them. The output from *ctags* may be fooled and incorrect for various constructs.

EXAMPLES

None.

RATIONALE

The option list was significantly reduced from that provided by historical implementations. The **−F** option was omitted as redundant, since it is the default. The **−B** option was omitted as being of very limited usefulness. The **−t** option was omitted since the recognition of **typedef**s is now required for C source files. The **−u** option was omitted because the update function was judged to be not only inefficient, but also rarely needed.

An early proposal included a **−w** option to suppress warning diagnostics. Since the types of such diagnostics could not be described, the option was omitted as being not useful.

The text for *LC_CTYPE* about compatibility with the C locale acknowledges that the ISO C standard imposes requirements on the locale used to process C source. This could easily be a superset of that known as ``the C locale'' by way of implementation extensions, or one of a few alternative locales for systems supporting different codesets.

The collation sequence of the tags file is not affected by *LC_COLLATE* because it is typically not used by human readers, but only by programs such as *vi* to locate the tag within the source files. Using the POSIX locale eliminates some of the problems of coordinating locales between the *ctags* file creator and the *vi* file reader.

Historically, the tags file has been used only by *ex* and *vi*. However, the format of the tags file has been published to encourage other programs to use the tags in new ways. The format allows either patterns or line numbers to find the identifiers because the historical *vi* recognizes either. The *ctags* utility does not produce the format using line numbers because it is not useful following any source file changes that add or delete lines. The documented search patterns match historical practice. It should be noted that literal leading <circumflex> or trailing <dollar-sign> characters in the search pattern will only behave correctly if anchored to the beginning of the line or end of the line by an additional <circumflex> or <dollar-sign> character.

Historical implementations also understand the objects used by the languages FORTRAN, Pascal, and sometimes LISP, and they understand the C source output by *lex* and *yacc*. The *ctags* utility is not required to accommodate these languages, although implementors are encouraged to do so.

The following historical option was not specified, as *vgrind* is not included in this volume of POSIX.1-2024:

−v If the **−v** flag is given, an index of the form expected by *vgrind* is produced on the standard output. This listing contains the function name, filename, and page number (assuming 64-line pages). Since the output is sorted into lexicographic order, it may be desired to run the output through *sort* **−f**. Sample use:

```
ctags −v files | sort −f > index
vgrind −x index
```

The special treatment of the tag **main** makes the use of *ctags* practical in directories with more than one program.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

If this utility is directed to create a new directory entry that contains any bytes that have the

encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

c17, *vi*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The OUTPUT FILES section is changed to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

IEEE PASC Interpretation 1003.2 #168 is applied, changing ``create'' to ``write'' in the DESCRIPTION.

Issue 7

The *ctags* utility is no longer dependent on support for the User Portability Utilities option.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to behave as follows:

a. Report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

b. Disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1312 is applied, inserting a missing line break in the example commands in the RATIONALE section.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

NAME

cut — cut out selected fields of each line of a file

SYNOPSIS

```
cut -b list [-n] [file...]
cut -c list [file...]
cut -f list [-d delim] [-s] [file...]
```

DESCRIPTION

The *cut* utility shall cut out bytes (**–b** option), characters (**–c** option), or character-delimited fields (**–f** option) from each line in one or more files, concatenate them, and write them to standard output.

OPTIONS

The *cut* utility shall conform to XBD Section 12.2 (on page 215).

The application shall ensure that the option-argument *list* (see options **–b**, **–c**, and **–f** below) is a <comma>-separated list or <blank>-separated list of positive numbers and ranges. Ranges can be in three forms. The first is two positive numbers separated by a <hyphen-minus> (*low–high*), which represents all fields from the first number to the second number. The second is a positive number preceded by a <hyphen-minus> (*–high*), which represents all fields from field number 1 to that number. The third is a positive number followed by a <hyphen-minus> (*low–*), which represents that number to the last field, inclusive. The elements in *list* can be repeated, can overlap, and can be specified in any order, but the bytes, characters, or fields selected shall be written in the order of the input data. If an element appears in the selection list more than once, it shall be written exactly once.

The following options shall be supported:

–b *list* Cut based on a *list* of bytes. Each selected byte shall be output unless the **–n** option is also specified. It shall not be an error to select bytes not present in the input line.

–c *list* Cut based on a *list* of characters. Each selected character shall be output. It shall not be an error to select characters not present in the input line.

–d *delim* Set the field delimiter to the character *delim*. The default is the <tab>.

–f *list* Cut based on a *list* of fields, assumed to be separated in the file by a delimiter character (see **–d**). Each selected field shall be output. Output fields shall be separated by a single occurrence of the field delimiter character. Lines with no field delimiters shall be passed through intact, unless **–s** is specified. It shall not be an error to select fields not present in the input line.

–n Do not split characters. When specified with the **–b** option, each element in *list* of the form *low–high* (<hyphen-minus>-separated numbers) shall be modified as follows:

- If the byte selected by *low* is not the first byte of a character, *low* shall be decremented to select the first byte of the character originally selected by *low*. If the byte selected by *high* is not the last byte of a character, *high* shall be decremented to select the last byte of the character prior to the character originally selected by *high*, or zero if there is no prior character. If the resulting range element has *high* equal to zero or *low* greater than *high*, the list element shall be dropped from *list* for that input line without causing an error.

Each element in *list* of the form *low–* shall be treated as above with *high* set to the

number of bytes in the current line, not including the terminating <newline>. Each element in *list* of the form *–high* shall be treated as above with *low* set to 1. Each element in *list* of the form *num* (a single number) shall be treated as above with *low* set to *num* and *high* set to *num*.

–s Suppress lines with no delimiter characters, when used with the **–f** option. Unless specified, lines with no delimiters shall be passed through untouched.

OPERANDS

The following operand shall be supported:

file A pathname of an input file. If no *file* operands are specified, or if a *file* operand is `'–'`, the standard input shall be used.

STDIN

The standard input shall be used only if no *file* operands are specified, or if a *file* operand is `'–'`. See the INPUT FILES section.

INPUT FILES

The input files shall be text files, except that line lengths shall be unlimited.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *cut*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The *cut* utility output shall be a concatenation of the selected bytes, characters, or fields (one of the following):

```
"%s\n", <concatenation of bytes>
"%s\n", <concatenation of characters>
"%s\n", <concatenation of fields and field delimiters>
```

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All input files were output successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *cut* and *fold* utilities can be used to create text files out of files with arbitrary line lengths. The *cut* utility should be used when the number of lines (or records) needs to remain constant. The *fold* utility should be used when the contents of long lines need to be kept contiguous.

Earlier versions of the *cut* utility worked in an environment where bytes and characters were considered equivalent (modulo <backspace> and <tab> processing in some implementations). In the extended world of multi-byte characters, the new **–b** option has been added. The **–n** option (used with **–b**) allows it to be used to act on bytes rounded to character boundaries. The algorithm specified for **–n** guarantees that:

```
cut -b 1-500 -n file > file1
cut -b 501- -n file > file2
```

ends up with all the characters in **file** appearing exactly once in **file1** or **file2**. (There is, however, a <newline> in both **file1** and **file2** for each <newline> in **file**.)

EXAMPLES

Examples of the option qualifier list:

1,4,7 Select the first, fourth, and seventh bytes, characters, or fields and field delimiters.

1–3,8 Equivalent to 1,2,3,8.

–5,10 Equivalent to 1,2,3,4,5,10.

3– Equivalent to third to last, inclusive.

The *low–high* forms are not always equivalent when used with **–b** and **–n** and multi-byte characters; see the description of **–n**.

The following command:

```
cut -d : -f 1,6 /etc/passwd
```

reads the System V password file (user database) and produces lines of the form:

```
<user ID>:<home directory>
```

Most utilities in this volume of POSIX.1-2024 work on text files. The *cut* utility can be used to turn files with arbitrary line lengths into a set of text files containing the same data. The *paste* utility can be used to create (or recreate) files with arbitrary line lengths. For example, if **file** contains long lines:

```
cut -b 1-500 -n file > file1
cut -b 501- -n file > file2
```


creates **file1** (a text file) with lines no longer than 500 bytes (plus the <newline>) and **file2** that contains the remainder of the data from **file**. (Note that **file2** is not a text file if there are lines in **file** that are longer than 500 + {LINE_MAX} bytes.) The original file can be recreated from **file1** and **file2** using the command:

```
paste -d "\0" file1 file2 > file
```

RATIONALE

Some historical implementations do not count <backspace> characters in determining character counts with the **-c** option. This may be useful for using *cut* for processing *nroff* output. It was deliberately decided not to have the **-c** option treat either <backspace> or <tab> characters in any special fashion. The *fold* utility does treat these characters specially.

Unlike other utilities, some historical implementations of *cut* exit after not finding an input file, rather than continuing to process the remaining *file* operands. This behavior is prohibited by this volume of POSIX.1-2024, where only the exit status is affected by this problem.

The behavior of *cut* when provided with either mutually-exclusive options or options that do not work logically together has been deliberately left unspecified in favor of global wording in Section 1.4 (on page 2462).

The OPTIONS section was changed in response to IEEE PASC Interpretation 1003.2 #149. The change represents historical practice on all known systems. The original standard was ambiguous on the nature of the output.

The *list* option-arguments are historically used to select the portions of the line to be written, but do not affect the order of the data. For example:

```
echo abcdefghi | cut -c6,2,4-7,1
```

yields `"abdefg"`.

A proposal to enhance *cut* with the following option:

-o Preserve the selected field order. When this option is specified, each byte, character, or field (or ranges of such) shall be written in the order specified by the *list* option-argument, even if this requires multiple outputs of the same bytes, characters, or fields.

was rejected because this type of enhancement is outside the scope of the IEEE P1003.2b draft standard.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 2478), *fold*, *grep*, *paste*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The OPTIONS section is changed to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

SD5-XCU-ERN-171 is applied, adding APPLICATION USAGE.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0080 [584] is applied.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

cxref — generate a C-language program cross-reference table (**DEVELOPMENT**)

SYNOPSIS

XSI
```
cxref [-cs] [-o file] [-w num] [-D name[=def]]... [-I dir]...
    [-U name]... file...
```

DESCRIPTION

The *cxref* utility shall analyze a collection of C-language *file*s and attempt to build a cross-reference table. Information from **#define** lines shall be included in the symbol table. A sorted listing shall be written to standard output of all symbols (auto, static, and global) in each *file* separately, or with the **–c** option, in combination. Each symbol shall contain an <asterisk> before the declaring reference.

OPTIONS

The *cxref* utility shall conform to XBD Section 12.2 (on page 215), except that the order of the **–D**, **–I**, and **–U** options (which are identical to their interpretation by *c17*) is significant. The following options shall be supported:

–c Write a combined cross-reference of all input files.

–s Operate silently; do not print input filenames.

–o *file* Direct output to named *file*.

–w *num* Format output no wider than *num* (decimal) columns. This option defaults to 80 if *num* is not specified or is less than 51.

–D Equivalent to *c17*.

–I Equivalent to *c17*.

–U Equivalent to *c17*.

OPERANDS

The following operand shall be supported:

file A pathname of a C-language source file.

STDIN

Not used.

INPUT FILES

The input files are C-language source files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *cxref*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the ordering of the output.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall be used for the cross-reference listing, unless the **−o** option is used to select a different output file.

The format of standard output is unspecified, except that the following information shall be included:

- If the **−c** option is not specified, each portion of the listing shall start with the name of the input file on a separate line.
- The name line shall be followed by a sorted list of symbols, each with its associated location pathname, the name of the function in which it appears (if it is not a function name itself), and line number references.
- Each line number may be preceded by an <asterisk> (`'*'`) flag, meaning that this is the declaring reference. Other single-character flags, with implementation-defined meanings, may be included.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

The output file named by the **−o** option shall be used instead of standard output.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

c17

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 5

In the SYNOPSIS, [**–U** *dir*] is changed to [**–U** *name*].

Issue 6

The APPLICATION USAGE section is added.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

date — write the date and time

SYNOPSIS

```
date [-u] [+format]
```

XSI

```
date [-u] mmddhhmm[[cc]yy]
```

DESCRIPTION

XSI The *date* utility shall write the date and time to standard output or attempt to set the system date and time. By default, the current date and time shall be written. If an operand beginning with '+' is specified, the output format of *date* shall be controlled by the conversion specifications and other text in the operand.

OPTIONS

The *date* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

−u Perform operations as if the *TZ* environment variable was set to the string "UTC0", or its equivalent historical value of "GMT0". Otherwise, *date* shall use the timezone indicated by the *TZ* environment variable or the system default if that variable is unset or null.

OPERANDS

The following operands shall be supported:

+format When the format is specified, the output shall be formatted as if by *strftime*( ) with the specified format string, and a *timeptr* argument that is the equivalent of *localtime*(&*now*) if **−u** is not specified or *gmtime*(&*now*) if **−u** is specified, where *now* is an object of type **time_t** containing the return value of *time*(0).

A <newline> shall always be appended to the output of *strftime*( ).

XSI *mmddhhmm*[[*cc*]*yy*]

Attempt to set the system date and time from the value given in the operand. This is only possible if the user has appropriate privileges and the system permits the setting of the system date and time. The first *mm* is the month (number); *dd* is the day (number); *hh* is the hour (number, 24-hour system); the second *mm* is the minute (number); *cc* is the century and is the first two digits of the year (this is optional); *yy* is the last two digits of the year and is optional. If century is not specified, then values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive. The current year is the default if *yy* is omitted.

Note: It is expected that in a future version of this standard the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *date*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

LC_TIME Determine the format and contents of date and time strings written by *date*.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

TZ Determine the timezone in which the time and date are written, unless the **−u** option is specified. If the *TZ* variable is unset or null and **−u** is not specified, an unspecified system default timezone is used.

ASYNCHRONOUS EVENTS

Default.

STDOUT

When no formatting operand is specified, the output in the POSIX locale shall be equivalent to specifying:

```
date "+%a %b %e %H:%M:%S %Z %Y"
```

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 The date was written successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Conversion specifiers are of unspecified format when not in the POSIX locale. Some of them can contain <newline> characters in some locales, so it may be difficult to use the format shown in standard output for parsing the output of *date* in those locales.

Since the default *date* utility format for locales other than the POSIX or C locale is not required to include anything beyond the date and time, whereas for the POSIX or C locale it also includes the day name and time zone, it may be necessary to specify a format (or override the locale-selection environment variables) to ensure this information is included when desired.

The range of values for `%S` extends from 0 to 60 seconds to accommodate the occasional leap second.

Although certain of the conversion specifiers in the POSIX locale (such as the name of the month) are shown with initial capital letters, this need not be the case in other locales. Programs using these fields may need to adjust the capitalization if the output is going to be used at the beginning of a sentence.

The date string formatting capabilities are intended for use in Gregorian-style calendars, possibly with a different starting year (or years). The `%x` and `%c` conversion specifications, however, are intended for local representation; these may be based on a different, non-Gregorian calendar.

The `%C` conversion specification was introduced to allow a fallback for the `%EC` (alternative year format base year); it can be viewed as the base of the current subdivision in the Gregorian calendar. The century number is calculated as the year divided by 100 and truncated to an integer; it should not be confused with the use of ordinal numbers for centuries (for example, ``twenty-first century''.) Both the `%Ey` and `%y` can then be viewed as the offset from `%EC` and `%C`, respectively.

The `E` and `O` modifiers modify the traditional conversion specifiers, so that they can always be used, even if the implementation (or the current locale) does not support the modifier.

The `E` modifier supports alternative date formats, such as the Japanese Emperor's Era, as long as these are based on the Gregorian calendar system. Extending the `E` modifiers to other date elements may provide an implementation-defined extension capable of supporting other calendar systems, especially in combination with the `O` modifier.

The `O` modifier supports time and date formats using the locale's alternative numerical symbols, such as Kanji or Hindi digits or ordinal number representation.

Non-European locales, whether they use Latin digits in computational items or not, often have local forms of the digits for use in date formats. This is not totally unknown even in Europe; a variant of dates uses Roman numerals for the months: the third day of September 1991 would be written as 3.IX.1991. In Japan, Kanji digits are regularly used for dates; in Arabic-speaking countries, Hindi digits are used. The `%d`, `%e`, `%H`, `%I`, `%m`, `%S`, `%U`, `%w`, `%W`, and `%y` conversion specifications always return the date and time field in Latin digits (that is, 0 to 9). The `%O` modifier was introduced to support the use for display purposes of non-Latin digits. In the *LC_TIME* category in *localedef*, the optional **alt_digits** keyword is intended for this purpose. As an example, assume the following (partial) *localedef* source:

```
alt_digits  "";"I";"II";"III";"IV";"V";"VI";"VII";"VIII" \
            "IX";"X";"XI";"XII"
d_fmt       "%e.%Om.%Y"
```

With the above date, the command:

```
date "+%x"
```


would yield 3.IX.1991. With the same **d_fmt**, but without the **alt_digits**, the command would yield 3.9.1991.

EXAMPLES

1. The following are input/output examples of *date* used at arbitrary times in the POSIX locale:

```
$ date
Tue Jun 26 09:58:10 PDT 1990

$ date "+DATE: %m/%d/%y%nTIME: %H:%M:%S"
DATE: 11/02/91
TIME: 13:36:16

$ date "+TIME: %r"
TIME: 01:36:32 PM
```

2. Examples for Denmark, where the default date and time format is `%a %d %b %Y %T %Z`:

```
$ LANG=da_DK.iso_8859-1 date
ons 02 okt 1991 15:03:32 CET

$ LANG=da_DK.iso_8859-1 \
    date "+DATO: %A den %e. %B %Y%nKLOKKEN: %H:%M:%S"
DATO: onsdag den 2. oktober 1991
KLOKKEN: 15:03:56
```

3. Examples for Germany, where the default date and time format is `%a %d.%h.%Y, %T %Z`:

```
$ LANG=De_DE.88591 date
Mi 02.Okt.1991, 15:01:21 MEZ

$ LANG=De_DE.88591 date "+DATUM: %A, %d. %B %Y%nZEIT: %H:%M:%S"
DATUM: Mittwoch, 02. Oktober 1991
ZEIT: 15:02:02
```

4. Examples for France, where the default date and time format is `%a %d %h %Y %Z %T`:

```
$ LANG=Fr_FR.88591 date
Mer 02 oct 1991 MET 15:03:32

$ LANG=Fr_FR.88591 date "+JOUR: %A %d %B %Y%nHEURE: %H:%M:%S"
JOUR: Mercredi 02 octobre 1991
HEURE: 15:03:56
```

RATIONALE

Some of the new options for formatting are from the ISO C standard. The **−u** option was introduced to allow portable access to Coordinated Universal Time (UTC). The string `"GMT0"` is allowed as an equivalent *TZ* value to be compatible with all of the systems using the BSD implementation, where this option originated.

The `%e` format conversion specification (adopted from System V) was added because the ISO C standard conversion specifications did not provide any way to produce the historical default *date* output during the first nine days of any month.

There are two varieties of day and week numbering supported (in addition to any others created with the locale-dependent `%E` and `%O` modifier characters):

- The historical variety in which Sunday is the first day of the week and the weekdays preceding the first Sunday of the year are considered week 0. These are represented by `%w` and `%U`. A variant of this is `%W`, using Monday as the first day of the week, but still referring to week 0. This view of the calendar was retained because so many historical applications depend on it and the ISO C standard *strftime*( ) function, on which many *date* implementations are based, was defined in this way.
- The international standard, based on the ISO 8601-1:2019 standard where Monday is the first weekday and the algorithm for the first week number is more complex: If the week (Monday to Sunday) containing January 1 has four or more days in the new year, then it is week 1; otherwise, it is week 53 of the previous year, and the next week is week 1. These are represented by the new conversion specifications `%u` and `%V`, added as a result of international comments.

Although this standard only requires the default *date* utility format, for locales other than the POSIX or C locale, to include the date and time, it is common for implementations to include day name and time zone information as well. (For the POSIX locale this is required, with the day name in `%a` format at the beginning and the time zone in `%Z` format before the year.) Implementations are encouraged to include the day name (in `%a` or `%A` format) and the time zone (in `%Z` or `%z` format) in the default *date* utility format for all of the locales they provide.

Some implementations have a **date_fmt** locale keyword (see Section 7.3.5, on page 152) as an extension, to specify the default *date* utility format for each locale. On such implementations, if the *localedef* utility is used to create a locale that does not have this information, the *date* utility must by default still produce output for that locale that includes both the time and the date.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD Section 7.3.5 (on page 152), Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *fprintf*( ), *strftime*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

Changes are made for Year 2000 alignment.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The `%EX` modified conversion specification is added.

The Open Group Corrigendum U048/2 is applied, correcting the examples.

The DESCRIPTION is updated to refer to conversion specifications, instead of field descriptors for consistency with the *LC_TIME* category.

A clarification is made such that the current year is the default if the *yy* argument is omitted when setting the system date and time.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/19 is applied, correcting the CHANGE HISTORY section.

Issue 8

Austin Group Defect 466 is applied, replacing the list of conversion specifications for the *+format* operand with a requirement that the output is formatted as if by a call to *strftime*( ) with specific arguments.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1345 is applied, adding paragraphs to APPLICATION USAGE and RATIONALE about the default *date* utility format.

NAME

dd — convert and copy a file

SYNOPSIS

```
dd [operand...]
```

DESCRIPTION

The *dd* utility shall copy the specified input file to the specified output file with possible conversions using specific input and output block sizes. It shall read the input one block at a time, using the specified input block size; it shall then process the block of data actually returned, which could be smaller than the requested block size. It shall apply any conversions that have been specified and write the resulting data to the output in blocks of the specified output block size. If the **bs**=*expr* operand is specified and no conversions other than **sync**, **noerror**, or **notrunc** are requested, the data returned from each input block shall be written as a separate output block; if the read returns less than a full block and the **sync** conversion is not specified, the resulting output block shall be the same size as the input block. If the **bs**=*expr* operand is not specified, or a conversion other than **sync**, **noerror**, or **notrunc** is requested, the input shall be processed and collected into full-sized output blocks until the end of the input is reached.

The processing order shall be as follows:

1. An input block is read. If the **iflags**=fullblock operand is specified, this might entail multiple reads; otherwise, the input block shall be used even if the read was shorter than the specified block size.
2. If the input block is shorter than the specified input block size and the **sync** conversion is specified, null bytes shall be appended to the input data up to the specified size. (If either **block** or **unblock** is also specified, <space> characters shall be appended instead of null bytes.) The remaining conversions and output shall include the pad characters as if they had been read from the input.
3. If the **bs**=*expr* operand is specified and no conversion other than **sync** or **noerror** is requested, the resulting data shall be written to the output as a single block, and the remaining steps are omitted.
4. If the **swab** conversion is specified, each pair of input data bytes shall be swapped. If there is an odd number of bytes in the input block, the last byte in the input record shall not be swapped.
5. Any remaining conversions (**block**, **unblock**, **lcase**, and **ucase**) shall be performed. These conversions shall operate on the input data independently of the input blocking; an input or output fixed-length record may span block boundaries.
6. The data resulting from input or conversion or both shall be aggregated into output blocks of the specified size. After the end of input is reached, any remaining output shall be written as a block without padding if **conv**=**sync** is not specified; thus, the final output block may be shorter than the output block size.

OPTIONS

None.

OPERANDS

All of the operands shall be processed before any input is read. The following operands shall be supported:

if=*file* Specify the input pathname; the default is standard input.

of=*file* Specify the output pathname; the default is standard output. If the **seek**=*expr* conversion is not also specified, the output file shall be truncated before the copy begins if an explicit **of**=*file* operand is specified, unless **conv**=**notrunc** is specified. If **seek**=*expr* is specified, but **conv**=**notrunc** is not, the effect of the copy shall be to preserve the blocks in the output file over which *dd* seeks, but no other portion of the output file shall be preserved. (If the size of the seek plus the size of the input file is less than the previous size of the output file, the output file shall be shortened by the copy. If the input file is empty and either the size of the seek is greater than the previous size of the output file or the output file did not previously exist, the size of the output file shall be set to the file offset after the seek.)

ibs=*expr* Specify the input block size, in bytes, by *expr* (default is 512).

obs=*expr* Specify the output block size, in bytes, by *expr* (default is 512).

bs=*expr* Set both input and output block sizes to *expr* bytes, superseding **ibs**= and **obs**=. If no conversion other than **sync**, **noerror**, and **notrunc** is specified, each input block shall be copied to the output as a single block without aggregating short blocks.

cbs=*expr* Specify the conversion block size for **block** and **unblock** in bytes by *expr* (default is zero). If **cbs**= is omitted or given a value of zero, using **block** or **unblock** produces unspecified results.

XSI The application shall ensure that this operand is also specified if the **conv**= operand is specified with a value of **ascii**, **ebcdic**, or **ibm**. For a **conv**= operand with an **ascii** value, the input is handled as described for the **unblock** value, except that characters are converted to ASCII before any trailing <space> characters are deleted. For **conv**= operands with **ebcdic** or **ibm** values, the input is handled as described for the **block** value except that the characters are converted to EBCDIC or IBM EBCDIC, respectively, after any trailing <space> characters are added.

skip=*n* Skip *n* input blocks (using the specified input block size) before starting to copy. On seekable files, the implementation shall read the blocks or seek past them; on non-seekable files, the blocks shall be read and the data shall be discarded.

seek=*n* Skip *n* blocks (using the specified output block size) from the beginning of the output file before copying. On non-seekable files, existing blocks shall be read and space from the current end-of-file to the specified offset, if any, filled with null bytes; on seekable files, the implementation shall seek to the specified offset or read the blocks as described for non-seekable files.

count=*n* Copy only *n* input blocks. If *n* is zero, it is unspecified whether no blocks or all blocks are copied.

conv=*value*[,*value* ...]
Where *value*s are <comma>-separated symbols from the following list:

XSI **ascii** Convert EBCDIC to ASCII; see Table 3-8 (on page 2780).

XSI **ebcdic** Convert ASCII to EBCDIC; see Table 3-8 (on page 2780).

XSI **ibm** Convert ASCII to a different EBCDIC set; see Table 3-9 (on page 2781).

XSI The **ascii**, **ebcdic**, and **ibm** values are mutually-exclusive.

block Treat the input as a sequence of <newline>-terminated or end-of-file-terminated variable-length records independent of the input block boundaries. Each record shall be converted to a record with a fixed length specified by the conversion block size. Any <newline> shall be removed from the input line; <space> characters shall be appended to lines that are shorter than their conversion block size to fill the block. Lines that are longer than the conversion block size shall be truncated to the largest number of characters that fit into that size; the number of truncated lines shall be reported (see the STDERR section).

The **block** and **unblock** values are mutually-exclusive.

unblock Convert fixed-length records to variable length. Read a number of bytes equal to the conversion block size (or the number of bytes remaining in the input, if less than the conversion block size), delete all trailing <space> characters, and append a <newline>.

lcase Map uppercase characters specified by the *LC_CTYPE* keyword **tolower** to the corresponding lowercase character. Characters for which no mapping is specified shall not be modified by this conversion.

The **lcase** and **ucase** symbols are mutually-exclusive.

ucase Map lowercase characters specified by the *LC_CTYPE* keyword **toupper** to the corresponding uppercase character. Characters for which no mapping is specified shall not be modified by this conversion.

swab Swap every pair of input bytes.

noerror Do not stop processing on an input error. When an input error occurs, a diagnostic message shall be written on standard error, followed by the current input and output block counts in the same format as used at completion (see the STDERR section). If the **sync** conversion is specified, the missing input shall be replaced with null bytes and processed normally; otherwise, the input block shall be omitted from the output.

notrunc Do not truncate the output file. Preserve blocks in the output file not explicitly written by this invocation of the *dd* utility. (See also the preceding **of**=*file* operand.)

sync Pad every input block to the size of the **ibs**= buffer, appending null bytes. (If either **block** or **unblock** is also specified, append <space> characters, rather than null bytes.)

iflags=fullblock

Perform as many reads as required to reach the full input block size or end of file, rather than acting on partial reads. If this operand is in effect, then the **count**= operand refers to the number of full input blocks rather than reads. The behavior is unspecified if **iflags**=fullblock is requested alongside the **sync**, **block**, or **unblock** conversions.

The behavior is unspecified if operands other than **conv**= are specified more than once.

For the **bs**=, **cbs**=, **ibs**=, and **obs**= operands, the application shall supply an expression specifying a size in bytes. The expression, *expr*, can be:

1. A positive decimal number
2. A positive decimal number followed by *k*, specifying multiplication by 1 024
3. A positive decimal number followed by *b*, specifying multiplication by 512
4. Two or more positive decimal numbers (with or without *k* or *b*) separated by *x*, specifying the product of the indicated values

All of the operands are processed before any input is read.

XSI The following two tables display the octal number character values used for the **ascii** and **ebcdic** conversions (first table) and for the **ibm** conversion (second table). In both tables, the ASCII values are the row and column headers and the EBCDIC values are found at their intersections. For example, ASCII 0012 (LF) is the second row, third column, yielding 0045 in EBCDIC. The inverted tables (for EBCDIC to ASCII conversion) are not shown, but are in one-to-one correspondence with these tables. The differences between the two tables are highlighted by small boxes drawn around five entries.

92033

Table 3-8 ASCII to EBCDIC Conversion

	0	1	2	3	4	5	6	7
0000	0000 NUL	0001 SOH	0002 STX	0003 ETX	0067 EOT	0055 ENQ	0056 ACK	0057 BEL
0010	0026 BS	0005 HT	0045 LF	0013 VT	0014 FF	0015 CR	0016 SO	0017 SI
0020	0020 DLE	0021 DC1	0022 DC2	0023 DC3	0074 DC4	0075 NAK	0062 SYN	0046 ETB
0030	0030 CAN	0031 EM	0077 SUB	0047 ESC	0034 IFS	0035 IGS	0036 IRS	0037 ITB
0040	0100 Sp	0132 !	0177 "	0173 #	0133 $	0154 %	0120 &	0175 '
0050	0115 (	0135)	0134 *	0116 +	0153 ,	0140 -	0113 .	0141 /
0060	0360 0	0361 1	0362 2	0363 3	0364 4	0365 5	0366 6	0367 7
0070	0370 8	0371 9	0172 :	0136 ;	0114 <	0176 =	0156 >	0157 ?
0100	0174 @	0301 A	0302 B	0303 C	0304 D	0305 E	0306 F	0307 G
0110	0310 H	0311 I	0321 J	0322 K	0323 L	0324 M	0325 N	0326 O
0120	0327 P	0330 Q	0331 R	0342 S	0343 T	0344 U	0345 V	0346 W
0130	0347 X	0350 Y	0351 Z	0255 [	0340 \	0275]	0232	0155 _
0140	0171 `	0201 a	0202 b	0203 c	0204 d	0205 e	0206 f	0207 g
0150	0210 h	0211 i	0221 j	0222 k	0223 l	0224 m	0225 n	0226 o
0160	0227 p	0230 q	0231 r	0242 s	0243 t	0244 u	0245 v	0246 w
0170	0247 x	0250 y	0251 z	0300 {	0117 |	0320 }	0137 ¬	0007 DEL
0200	0040 DS	0041 SOS	0042 FS	0043 WUS	0044 BYP	0025 NL	0006 RNL	0027 POC
0210	0050 SA	0051 SFE	0052 SM	0053 CSP	0054 MFA	0011 SPS	0012 RPT	0033 CU1
0220	0060	0061	0032 UBS	0063 IR	0064 PP	0065 TRN	0066 NBS	0010 GE
0230	0070 SBS	0071 IT	0072 RFF	0073 CU3	0004 SEL	0024 RES	0076	0341
0240	0101	0102	0103	0104	0105	0106	0107	0110
0250	0111	0121	0122	0123	0124	0125	0126	0127
0260	0130	0131	0142	0143	0144	0145	0146	0147
0270	0150	0151	0160	0161	0162	0163	0164	0165
0300	0166	0167	0170	0200	0212	0213	0214	0215
0310	0216	0217	0220	0152 ¦	0233	0234	0235	0236
0320	0237	0240	0252	0253	0254	0112 ¢	0256	0257
0330	0260	0261	0262	0263	0264	0265	0266	0267
0340	0270	0271	0272	0273	0274	0241	0276	0277
0350	0312	0313	0314 ⌐	0315	0316 ↵	0317	0332	0333
0360	0334	0335	0336	0337	0352	0353	0354 ⊣	0355
0370	0356	0357	0372 |	0373	0374	0375	0376	0377 EO

92034

Table 3-9 ASCII to IBM EBCDIC Conversion

	0	1	2	3	4	5	6	7
0000	0000 NUL	0001 SOH	0002 STX	0003 ETX	0067 EOT	0055 ENQ	0056 ACK	0057 BEL
0010	0026 BS	0005 HT	0045 LF	0013 VT	0014 FF	0015 CR	0016 SO	0017 SI
0020	0020 DLE	0021 DC1	0022 DC2	0023 DC3	0074 DC4	0075 NAK	0062 SYN	0046 ETB
0030	0030 CAN	0031 EM	0077 SUB	0047 ESC	0034 IFS	0035 IGS	0036 IRS	0037 ITB
0040	0100 Sp	0132 !	0177 "	0173 #	0133 $	0154 %	0120 &	0175 '
0050	0115 (	0135)	0134 *	0116 +	0153 ,	0140 -	0113 .	0141 /
0060	0360 0	0361 1	0362 2	0363 3	0364 4	0365 5	0366 6	0367 7
0070	0370 8	0371 9	0172 :	0136 ;	0114 <	0176 =	0156 >	0157 ?
0100	0174 @	0301 A	0302 B	0303 C	0304 D	0305 E	0306 F	0307 G
0110	0310 H	0311 I	0321 J	0322 K	0323 L	0324 M	0325 N	0326 O
0120	0327 P	0330 Q	0331 R	0342 S	0343 T	0344 U	0345 V	0346 W
0130	0347 X	0350 Y	0351 Z	0255 [	0340 \	0275]	0137 ¬	0155 _
0140	0171 `	0201 a	0202 b	0203 c	0204 d	0205 e	0206 f	0207 g
0150	0210 h	0211 i	0221 j	0222 k	0223 l	0224 m	0225 n	0226 o
0160	0227 p	0230 q	0231 r	0242 s	0243 t	0244 u	0245 v	0246 w
0170	0247 x	0250 y	0251 z	0300 {	0117 |	0320 }	0241	0007 DEL
0200	0040 DS	0041 SOS	0042 FS	0043 WUS	0044 BYP	0025 NL	0006 RNL	0027 POC
0210	0050 SA	0051 SFE	0052 SM	0053 CSP	0054 MFA	0011 SPS	0012 RPT	0033 CU1
0220	0060	0061	0032 UBS	0063 IR	0064 PP	0065 TRN	0066 NBS	0010 GE
0230	0070 SBS	0071 IT	0072 RFF	0073 CU3	0004 SEL	0024 RES	0076	0341
0240	0101	0102	0103	0104	0105	0106	0107	0110
0250	0111	0121	0122	0123	0124	0125	0126	0127
0260	0130	0131	0142	0143	0144	0145	0146	0147
0270	0150	0151	0160	0161	0162	0163	0164	0165
0300	0166	0167	0170	0200	0212	0213	0214	0215
0310	0216	0217	0220	0232	0233	0234	0235	0236
0320	0237	0240	0252	0253	0254	0255 [	0256	0257
0330	0260	0261	0262	0263	0264	0265	0266	0267
0340	0270	0271	0272	0273	0274	0275]	0276	0277
0350	0312	0313	0314 ⌐	0315	0316 ↑	0317	0332	0333
0360	0334	0335	0336	0337	0352	0353	0354 ⊣	0355
0370	0356	0357	0372 |	0373	0374	0375	0376	0377 EO

92035 **STDIN**

92036 If no **if**= operand is specified, the standard input shall be used. See the INPUT FILES section.

92037 **INPUT FILES**

92038 The input file can be any file type.

92039 **ENVIRONMENT VARIABLES**

92040 The following environment variables shall affect the execution of *dd*:

92041 *LANG* Provide a default value for the internationalization variables that are unset or null.
92042 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
92043 variables used to determine the values of locale categories.)

92044 *LC_ALL* If set to a non-empty string value, override the values of all the other
92045 internationalization variables.

92046 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
92047 characters (for example, single-byte as opposed to multi-byte characters in
92048 arguments and input files), the classification of characters as uppercase or
92049 lowercase, and the mapping of characters from one case to the other.

92050 *LC_MESSAGES*
92051 Determine the locale that should be used to affect the format and contents of
92052 diagnostic messages written to standard error and informative messages written to
92053 standard output.

92054 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

92055 **ASYNCHRONOUS EVENTS**

92056 For SIGINT, the *dd* utility shall interrupt its current processing, write status information to
92057 standard error, and terminate abnormally as if by the default action for SIGINT. One or more
92058 implementation defined non-job-control signals other than SIGABRT, SIGHUP, and SIGTERM
92059 may write status information to standard error and continue processing. All other signals
92060 (including job control signals, SIGABRT, SIGHUP, and SIGTERM) shall take their default action;
92061 see the ASYNCHRONOUS EVENTS section in Section 1.4 (on page 2462).

92062 **STDOUT**

92063 If no **of**= operand is specified, the standard output shall be used. The nature of the output
92064 depends on the operands selected.

92065 **STDERR**

92066 On completion, *dd* shall write the number of input and output blocks to standard error. In the
92067 POSIX locale the following formats shall be used:

92068 `"%u+%u records in\n", <number of whole input blocks>,`
92069 `    <number of partial input blocks>`

92070 `"%u+%u records out\n", <number of whole output blocks>,`
92071 `    <number of partial output blocks>`

92072 A partial input block is one for which *read*() returned less than the input block size. A partial
92073 output block is one that was written with fewer bytes than specified by the output block size.

92074 In addition, when there is at least one truncated block, the number of truncated blocks shall be
92075 written to standard error. In the POSIX locale, the format shall be:

92076 `"%u truncated %s\n", <number of truncated blocks>, "record" (if`
92077 `    <number of truncated blocks> is one) "records" (otherwise)`

92078 Diagnostic messages may also be written to standard error.

92079 **OUTPUT FILES**

92080 If the **of**= operand is used, the output shall be the same as described in the STDOUT section.

92081 **EXTENDED DESCRIPTION**

92082 None.

92083 **EXIT STATUS**

92084 The following exit values shall be returned:

92085 0 Successful completion.

92086 >0 An error occurred.

92087 **CONSEQUENCES OF ERRORS**

92088 If an input error is detected and the **noerror** conversion has not been specified, any partial
92089 output block shall be written to the output file, a diagnostic message shall be written, and the
92090 copy operation shall be discontinued. If some other error is detected, a diagnostic message shall
92091 be written and the copy operation shall be discontinued.

92092 **APPLICATION USAGE**

92093 The input and output block size can be specified to take advantage of raw physical I/O.

92094 There are many different versions of the EBCDIC codesets. The ASCII and EBCDIC conversions
92095 specified for the *dd* utility perform conversions for the version specified by the tables.

92096 Using the **count**= operand of *dd* with a pipe or FIFO as the input can lead to surprising results,
92097 since these file types are prone to encountering short reads for any input block size other than 1.
92098 Unless the **iflags**=fullblock operand is in effect, *dd* will stop after the specified number of reads,
92099 rather than full input blocks, and therefore can often result in fewer bytes being output than the
92100 product of the count and input block size.

92101 **EXAMPLES**

92102 The following command:

92103 ```
dd if=/dev/rmt0h of=/dev/rmt1h
```

92104 copies from tape drive 0 to tape drive 1, using a common historical device naming convention.

92105 The following command:

92106 ```
dd ibs=10  skip=1
```

92107 strips the first 10 bytes from standard input.

92108 This example reads an EBCDIC tape blocked ten 80-byte EBCDIC card images per block into the
92109 ASCII file **x**:

92110 ```
dd if=/dev/tape of=x ibs=800 cbs=80 conv=ascii,lcase
```

92111 **RATIONALE**

92112 The OPTIONS section is listed as ``None'' because there are no options recognized by historical
92113 *dd* utilities. Certainly, many of the operands could have been designed to use the Utility Syntax
92114 Guidelines, which would have resulted in the classic hyphenated option letters. In this version
92115 of this volume of POSIX.1-2024, *dd* retains its curious JCL-like syntax due to the large number of
92116 applications that depend on the historical implementation.

92117 A suggested implementation technique for **conv**=**noerror**,**sync** is to zero (or <space>-fill, if
92118 **block**ing or **unblock**ing) the input buffer before each read and to write the contents of the input
92119 buffer to the output even after an error. In this manner, any data transferred to the input buffer
92120 before the error was detected is preserved. Another point is that a failed read on a regular file or
92121 a disk generally does not increment the file offset, and *dd* must then seek past the block on which

  
```

92122 the error occurred; otherwise, the input error occurs repetitively. When the input is a magnetic
92123 tape, however, the tape normally has passed the block containing the error when the error is
92124 reported, and thus no seek is necessary.

92125 The default **ibs**= and **obs**= sizes are specified as 512 bytes because there are historical (largely
92126 portable) scripts that assume these values. If they were left unspecified, unusual results could
92127 occur if an implementation chose an odd block size.

92128 Historical implementations of *dd* used *creat*() when processing **of**=*file*. This makes the **seek**=
92129 operand unusable except on special files. The **conv**=**notrunc** feature was added because more
92130 recent BSD-based implementations use *open*() (without O_TRUNC) instead of *creat*(), but they
92131 fail to delete output file contents after the data copied.

92132 The *w* multiplier (historically meaning *word*), is used in System V to mean 2 and in 4.2 BSD to
92133 mean 4. Since *word* is inherently non-portable, its use is not supported by this volume of
92134 POSIX.1-2024.

92135 Standard EBCDIC does not have the characters '[' and ']'. The values used in the table are
92136 taken from a common print train that does contain them. Other than those characters, the print
92137 train values are not filled in, but appear to provide some of the motivation for the historical
92138 choice of translations reflected here.

92139 The Standard EBCDIC table provides a 1:1 translation for all 256 bytes.

92140 The IBM EBCDIC table does not provide such a translation. The marked cells in the tables differ
92141 in such a way that:

92142 1. EBCDIC 0112 ('¢') and 0152 (broken pipe) do not appear in the table.

92143 2. EBCDIC 0137 ('¬') translates to/from ASCII 0236 ('^'). In the standard table, EBCDIC
92144 0232 (no graphic) is used.

92145 3. EBCDIC 0241 ('~') translates to/from ASCII 0176 ('~'). In the standard table, EBCDIC
92146 0137 ('¬') is used.

92147 4. 0255 ('[') and 0275 (']') appear twice, once in the same place as for the standard table
92148 and once in place of 0112 ('¢') and 0241 ('~').

92149 In net result:

92150 EBCDIC 0275 (']') displaced EBCDIC 0241 ('~') in cell 0345.

92151 That displaced EBCDIC 0137 ('¬') in cell 0176.

92152 That displaced EBCDIC 0232 (no graphic) in cell 0136.

92153 That replaced EBCDIC 0152 (broken pipe) in cell 0313.

92154 EBCDIC 0255 ('[') replaced EBCDIC 0112 ('¢').

92155 This translation, however, reflects historical practice that (ASCII) '~' and '¬' were often
92156 mapped to each other, as were '[' and '¢'; and ']' and (EBCDIC) '~'.

92157 The **cbs** operand is required if any of the **ascii**, **ebcdic**, or **ibm** operands are specified. For the
92158 **ascii** operand, the input is handled as described for the **unblock** operand except that characters
92159 are converted to ASCII before the trailing <space> characters are deleted. For the **ebcdic** and
92160 **ibm** operands, the input is handled as described for the **block** operand except that the characters
92161 are converted to EBCDIC or IBM EBCDIC after the trailing <space> characters are added.

92162 The **block** and **unblock** keywords are from historical BSD practice.

The consistent use of the word **record** in standard error messages matches most historical practice. An earlier version of System V used **block**, but this has been updated in more recent releases.

Early proposals only allowed two numbers separated by **x** to be used in a product when specifying **bs**=, **cbs**=, **ibs**=, and **obs**= sizes. This was changed to reflect the historical practice of allowing multiple numbers in the product as provided by Version 7 and all releases of System V and BSD.

A change to the **swab** conversion is required to match historical practice and is the result of IEEE PASC Interpretations 1003.2 #03 and #04, submitted for the ISO POSIX-2: 1993 standard.

A change to the handling of SIGINT is required to match historical practice and is the result of IEEE PASC Interpretation 1003.2 #06 submitted for the ISO POSIX-2: 1993 standard.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

A future version of this standard may introduce the SIGINFO signal; on platforms where such a signal is available, it is recommended that this signal be used for reporting status without terminating the process.

SEE ALSO

Section 1.4 (on page 2462), *sed*, *tr*

XBD Chapter 8 (on page 167)

CHANGE HISTORY

First released in Issue 2.

Issue 5

The second paragraph of the **cbs**= description is reworded and marked EX.

The FUTURE DIRECTIONS section is added.

Issue 6

Changes are made to **swab** conversion and SIGINT handling to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

IEEE PASC Interpretation 1003.2 #209 is applied, clarifying the interaction between *dd* **of**=*file* and **conv**=**notrunc**.

Issue 7

Austin Group Interpretation 1003.1-2001 #102 is applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0081 [907] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 406 is applied, adding the **iflags**=fullblock operand.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1159 is applied, changing the ASYNCHRONOUS EVENTS and FUTURE

DIRECTIONS sections.

Austin Group Defect 1497 is applied, changing the EXIT STATUS section.

92207 **NAME**

92208 delta — make a delta (change) to an SCCS file (**DEVELOPMENT**)

92209 **SYNOPSIS**

92210 XSI
```
delta [-nps] [-g list] [-m mrlist] [-r SID] [-y[comment]] file...
```

92211 **DESCRIPTION**

92212 The *delta* utility shall be used to permanently introduce into the named SCCS files changes that
92213 were made to the files retrieved by *get* (called the *g-files*, or generated files).

92214 **OPTIONS**

92215 The *delta* utility shall conform to XBD Section 12.2 (on page 215), except that the –**y** option has an
92216 optional option-argument. This optional option-argument shall not be presented as a separate
92217 argument.

92218 The following options shall be supported:

92219 –**r** *SID* Uniquely identify which delta is to be made to the SCCS file. The use of this option
92220 shall be necessary only if two or more outstanding *get* commands for editing (*get*
92221 –**e**) on the same SCCS file were done by the same person (login name). The SID
92222 value specified with the –**r** option can be either the SID specified on the *get*
92223 command line or the SID to be made as reported by the *get* utility; see *get* (on page
92224 2964).

92225 –**s** Suppress the report to standard output of the activity associated with each *file*. See
92226 the STDOUT section.

92227 –**n** Specify retention of the edited *g-file* (normally removed at completion of delta
92228 processing).

92229 –**g** *list* Specify a *list* (see *get* for the definition of *list*) of deltas that shall be ignored when
92230 the file is accessed at the change level (SID) created by this delta.

92231 –**m** *mrlist* Specify a modification request (MR) number that the application shall supply as
92232 the reason for creating the new delta. This shall be used if the SCCS file has the **v**
92233 flag set; see *admin*.

92234 If –**m** is not used and `'-'` is not specified as a file argument, and the standard
92235 input is a terminal, the prompt described in the STDOUT section shall be written
92236 to standard output before the standard input is read; if the standard input is not a
92237 terminal, no prompt shall be issued.

92238 MRs in a list shall be separated by <blank> characters or escaped <newline>
92239 characters. An unescaped <newline> shall terminate the MR list. The escape
92240 character is <backslash>.

92241 If the **v** flag has a value, it shall be taken to be the name of a program which
92242 validates the correctness of the MR numbers. If a non-zero exit status is returned
92243 from the MR number validation program, the *delta* utility shall terminate. (It is
92244 assumed that the MR numbers were not all valid.)

92245 –**y**[*comment*] Describe the reason for making the delta. The *comment* shall be an arbitrary group
92246 of lines that would meet the definition of a text file. Implementations shall support
92247 *comment*s from zero to 512 bytes and may support longer values. A null string
92248 (specified as either –**y**, –**y**`""`, or in response to a prompt for a comment) shall be
92249 considered a valid *comment*.

92250 If –**y** is not specified and `'-'` is not specified as a file argument, and the standard

92251 input is a terminal, the prompt described in the STDOUT section shall be written
92252 to standard output before the standard input is read; if the standard input is not a
92253 terminal, no prompt shall be issued. An unescaped <newline> shall terminate the
92254 comment text. The escape character is <backslash>.

92255 The **−y** option shall be required if the *file* operand is specified as `'−'`.

92256 **−p** Write (to standard output) the SCCS file differences before and after the delta is
92257 applied in *diff* format; see *diff*.

92258 **OPERANDS**

92259 The following operand shall be supported:

92260 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *delta*
92261 utility shall behave as though each file in the directory were specified as a named
92262 file, except that non-SCCS files (last component of the pathname does not begin
92263 with **s.**) and unreadable files shall be silently ignored.

92264 If exactly one *file* operand appears, and it is `'−'`, the standard input shall be read;
92265 each line of the standard input shall be taken to be the name of an SCCS file to be
92266 processed. Non-SCCS files and unreadable files shall be silently ignored.

92267 **STDIN**

92268 The standard input shall be a text file used only in the following cases:

92269 • To read an *mrlist* or a *comment* (see the **−m** and **−y** options).

92270 • A *file* operand shall be specified as `'−'`. In this case, the **−y** option needs to be used to
92271 specify the comment, and if the SCCS file has the **v** flag set, the **−m** option also needs to be
92272 used to specify the MR list.

92273 **INPUT FILES**

92274 Input files shall be text files whose data is to be included in the SCCS files. If the first character of
92275 any line of an input file is <SOH> in the POSIX locale, the results are unspecified. If this file
92276 contains more than 99 999 lines, the number of lines recorded in the header for this file shall be
92277 99 999 for this delta.

92278 **ENVIRONMENT VARIABLES**

92279 The following environment variables shall affect the execution of *delta*:

92280 *LANG* Provide a default value for the internationalization variables that are unset or null.
92281 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
92282 variables used to determine the values of locale categories.)

92283 *LC_ALL* If set to a non-empty string value, override the values of all the other
92284 internationalization variables.

92285 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
92286 characters (for example, single-byte as opposed to multi-byte characters in
92287 arguments and input files).

92288 *LC_MESSAGES*
92289 Determine the locale that should be used to affect the format and contents of
92290 diagnostic messages written to standard error, and informative messages written
92291 to standard output.

92292 *NLSPATH* Determine the location of messages objects and message catalogs.

92293 *TZ* Determine the timezone in which the time and date are written in the SCCS file. If
92294 the *TZ* variable is unset or NULL, an unspecified system default timezone is used.

92295 **ASYNCHRONOUS EVENTS**

92296 If SIGINT is caught, temporary files shall be cleaned up and *delta* shall exit with a non-zero exit
92297 code. The standard action shall be taken for all other signals; see Section 1.4 (on page 2462).

92298 **STDOUT**

92299 The standard output shall be used only for the following messages in the POSIX locale:

92300 • Prompts (see the –**m** and –**y** options) in the following formats:

92301 `"MRs? "`

92302 `"comments? "`

92303 The MR prompt, if written, shall always precede the comments prompt.

92304 • A report of each file's activities (unless the –**s** option is specified) in the following format:

92305 `"%s\n%d inserted\n%d deleted\n%d unchanged\n", <`*`New SID`*`>,`
92306 `<`*`number of lines inserted`*`>, <`*`number of lines deleted`*`>,`
92307 `<`*`number of lines unchanged`*`>`

92308 **STDERR**

92309 The standard error shall be used only for diagnostic messages.

92310 **OUTPUT FILES**

92311 Any SCCS files updated shall be files of an unspecified format.

92312 **EXTENDED DESCRIPTION**

92313 **System Date and Time**

92314 When a delta is added to an SCCS file, the system date and time shall be recorded for the new
92315 delta. If a *get* is performed using an SCCS file with a date recorded apparently in the future, the
92316 behavior is unspecified.

92317 **EXIT STATUS**

92318 The following exit values shall be returned:

92319 0 Successful completion.

92320 >0 An error occurred.

92321 **CONSEQUENCES OF ERRORS**

92322 Default.

92323 **APPLICATION USAGE**

92324 Problems can arise if the system date and time have been modified (for example, put forward
92325 and then back again, or unsynchronized clocks across a network) and can also arise when
92326 different values of the *TZ* environment variable are used.

92327 Problems of a similar nature can also arise for the operation of the *get* utility, which records the
92328 date and time in the file body.

92329 **EXAMPLES**

92330 None.

92331 **RATIONALE**

92332 None.

92333 **FUTURE DIRECTIONS**

92334 If this utility is directed to display a pathname that contains any bytes that have the encoded
92335 value of a <newline> character when <newline> is a terminator or separator in the output
92336 format being used, implementations are encouraged to treat this as an error. A future version of
92337 this standard may require implementations to treat this as an error.

92338 If this utility is directed to create a new directory entry that contains any bytes that have the
92339 encoded value of a <newline> character, implementations are encouraged to treat this as an
92340 error. A future version of this standard may require implementations to treat this as an error.

92341 **SEE ALSO**

92342 Section 1.4 (on page 2462), *admin*, *diff*, *get*, *prs*, *rmdel*

92343 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

92344 **CHANGE HISTORY**

92345 First released in Issue 2.

92346 **Issue 5**

92347 The output format description in the STDOUT section is corrected.

92348 **Issue 6**

92349 The APPLICATION USAGE section is added.

92350 The normative text is reworded to avoid use of the term ``must'' for application requirements.

92351 The Open Group Base Resolution bwg2001-007 is applied as follows:

92352 • The use of `'-'` as a file argument is clarified.

92353 • The use of STDIN is added.

92354 • The ASYNCHRONOUS EVENTS section is updated to remove the implicit requirement
92355 that implementations re-signal themselves when catching a normally fatal signal.

92356 • New text is added to the INPUT FILES section warning that the maximum lines recorded
92357 in the file is 99 999.

92358 New text is added to the EXTENDED DESCRIPTION and APPLICATION USAGE sections
92359 regarding how the system date and time may be taken into account, and the *TZ* environment
92360 variable is added to the ENVIRONMENT VARIABLES section as per The Open Group Base
92361 Resolution bwg2001-007.

92362 **Issue 7**

92363 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

92364 **Issue 8**

92365 Austin Group Defect 251 is applied, encouraging implementations to behave as follows:

92366 a. Report an error if a utility is directed to display a pathname that contains any bytes that
92367 have the encoded value of a <newline> character when <newline> is a terminator or
92368 separator in the output format being used.

92369 b. Disallow the creation of filenames containing any bytes that have the encoded value of a
92370 <newline> character.

92371 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

92372 **NAME**

92373 df — report free disk space

92374 **SYNOPSIS**

92375 XSI `df [-k] [-P|-t] [file...]`

92376 **DESCRIPTION**

92377 XSI The *df* utility shall write the amount of available space and file slots for file systems on which
92378 the invoking user has appropriate read access. File systems shall be specified by the *file*
92379 operands; when none are specified, information shall be written for all file systems. The format
92380 of the default output from *df* is unspecified, but all space figures are reported in 512-byte units,
92381 unless the **–k** option is specified. This output shall contain at least the file system names, amount
92382 XSI of available space on each of these file systems, and, if no options other than **–t** are specified, the
92383 number of free file slots, or *inode*s, available; when **–t** is specified, the output shall contain the
92384 total allocated space as well.

92385 **OPTIONS**

92386 The *df* utility shall conform to XBD Section 12.2 (on page 215).

92387 The following options shall be supported:

92388 **–k** Use 1 024-byte units, instead of the default 512-byte units, when writing space
92389 figures.

92390 **–P** Produce output in the format described in the STDOUT section.

92391 XSI **–t** Include total allocated-space figures in the output.

92392 **OPERANDS**

92393 The following operand shall be supported:

92394 *file* A pathname of a file within the hierarchy of the desired file system. If a file other
92395 XSI than a FIFO, a regular file, a directory, or a special file representing the device
92396 containing the file system (for example, **/dev/dsk/0s1**) is specified, the results are
92397 unspecified. If the *file* operand names a file other than a special file containing a file
92398 system, *df* shall write the amount of free space in the file system containing the
92399 XSI specified *file* operand. Otherwise, *df* shall write the amount of free space in that
92400 file system.

92401 **STDIN**

92402 Not used.

92403 **INPUT FILES**

92404 None.

92405 **ENVIRONMENT VARIABLES**

92406 The following environment variables shall affect the execution of *df*:

92407 *LANG* Provide a default value for the internationalization variables that are unset or null.
92408 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
92409 variables used to determine the values of locale categories.)

92410 *LC_ALL* If set to a non-empty string value, override the values of all the other
92411 internationalization variables.

92412 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
92413 characters (for example, single-byte as opposed to multi-byte characters in
92414 arguments).

92415 *LC_MESSAGES*
92416 Determine the locale that should be used to affect the format and contents of
92417 diagnostic messages written to standard error and informative messages written to
92418 standard output.

92419 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

92420 **ASYNCHRONOUS EVENTS**
92421 Default.

92422 **STDOUT**
92423 When both the –**k** and –**P** options are specified, the following header line shall be written (in the
92424 POSIX locale):

92425 `"Filesystem 1024-blocks Used Available Capacity Mounted on\n"`

92426 When the –**P** option is specified without the –**k** option, the following header line shall be written
92427 (in the POSIX locale):

92428 `"Filesystem 512-blocks Used Available Capacity Mounted on\n"`

92429 The implementation may adjust the spacing of the header line and the individual data lines so
92430 that the information is presented in orderly columns.

92431 The remaining output with –**P** shall consist of one line of information for each specified file
92432 system. These lines shall be formatted as follows:

92433 `"%s %d %d %d %d%% %s\n", <file system name>, <total space>,`
92434 `<space used>, <space free>, <percentage used>,`
92435 `<file system root>`

92436 In the following list, all quantities expressed in 512-byte units (1 024-byte when –**k** is specified)
92437 shall be rounded up to the next higher unit. The fields are:

92438 <*file system name*>
92439 The name of the file system, in an implementation-defined format.

92440 <*total space*> The total size of the file system in 512-byte units. The exact meaning of this figure
92441 is implementation-defined, but should include <*space used*>, <*space free*>, plus any
92442 space reserved by the system not normally available to a user.

92443 <*space used*> The total amount of space allocated to existing files in the file system, in 512-byte
92444 units.

92445 <*space free*> The total amount of space available within the file system for the creation of new
92446 files by unprivileged users, in 512-byte units. When this figure is less than or equal
92447 to zero, it shall not be possible to create any new files on the file system without
92448 first deleting others, unless the process has appropriate privileges. The figure
92449 written may be less than zero.

92450 <*percentage used*>
92451 The percentage of the normally available space that is currently allocated to all files
92452 on the file system. This shall be calculated using the fraction:

92453 `<space used>/( <space used>+ <space free>)`

92454 expressed as a percentage. This percentage may be greater than 100 if <*space free*>
92455 is less than zero. The percentage value shall be expressed as a positive integer, with
92456 any fractional result causing it to be rounded to the next highest integer.

92457 <*file system root*>
92458 The directory below which the file system hierarchy appears.

92459 XSI The output format is unspecified when **–t** is used.

92460 **STDERR**
92461 The standard error shall be used only for diagnostic messages.

92462 **OUTPUT FILES**
92463 None.

92464 **EXTENDED DESCRIPTION**
92465 None.

92466 **EXIT STATUS**
92467 The following exit values shall be returned:

92468 0 Successful completion.

92469 >0 An error occurred.

92470 **CONSEQUENCES OF ERRORS**
92471 Default.

92472 **APPLICATION USAGE**
92473 On most systems, the ``name of the file system, in an implementation-defined format'' is the
92474 special file on which the file system is mounted.

92475 On large file systems, the calculation specified for percentage used can create huge rounding
92476 errors.

92477 **EXAMPLES**

92478 1. The following example writes portable information about the **/usr** file system:

92479 ```
df –P /usr
```

92480 2. Assuming that **/usr/src** is part of the **/usr** file system, the following produces the same
92481 output as the previous example:

92482 ```
df –P /usr/src
```

92483 **RATIONALE**
92484 The behavior of *df* with the **–P** option is the default action of the 4.2 BSD *df* utility. The uppercase
92485 **–P** was selected to avoid collision with a known industry extension using **–p**.

92486 Historical *df* implementations vary considerably in their default output. It was therefore
92487 necessary to describe the default output in a loose manner to accommodate all known historical
92488 implementations and to add a portable option (**–P**) to provide information in a portable format.

92489 The use of 512-byte units is historical practice and maintains compatibility with *ls* and other
92490 utilities in this volume of POSIX.1-2024. This does not mandate that the file system itself be
92491 based on 512-byte blocks. The **–k** option was added as a compromise measure. It was agreed by
92492 the standard developers that 512 bytes was the best default unit because of its complete
92493 historical consistency on System V (*versus* the mixed 512/1 024-byte usage on BSD systems), and
92494 that a **–k** option to switch to 1 024-byte units was a good compromise. Users who prefer the
92495 more logical 1 024-byte quantity can easily alias *df* to *df* **–k** without breaking many historical
92496 scripts relying on the 512-byte units.

92497 It was suggested that *df* and the various related utilities be modified to access a *BLOCKSIZE*
92498 environment variable to achieve consistency and user acceptance. Since this is not historical

practice on any system, it is left as a possible area for system extensions and will be re-evaluated in a future version if it is widely implemented.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

find

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is marked as part of the User Portability Utilities option.

Issue 7

Austin Group Interpretation 1003.1-2001 #099 is applied.

The *df* utility is removed from the User Portability Utilities option. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0082 [156] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

92525 **NAME**

92526 diff — compare two files

92527 **SYNOPSIS**

92528 `diff [-c|-e|-f|-u|-C n|-U n] [-br] file1 file2`

92529 **DESCRIPTION**

92530 The *diff* utility shall compare the contents of *file1* and *file2* and write to standard output a list of
92531 changes necessary to convert *file1* into *file2*. This list should be minimal. No output shall be
92532 produced if the files are identical.

92533 **OPTIONS**

92534 The *diff* utility shall conform to XBD Section 12.2 (on page 215).

92535 The following options shall be supported:

92536 **−b** Cause any amount of white space at the end of a line to be treated as a single
92537 <newline> (that is, the white-space characters preceding the <newline> are
92538 ignored) and other strings of white-space characters, not including <newline>
92539 characters, to compare equal.

92540 **−c** Produce output in a form that provides three lines of copied context.

92541 **−C** *n* Produce output in a form that provides *n* lines of copied context (where *n* shall be
92542 interpreted as a positive decimal integer).

92543 **−e** Produce output in a form suitable as input for the *ed* utility, which can then be used
92544 to convert *file1* into *file2*.

92545 **−f** Produce output in an alternative form, similar in format to **−e**, but not intended to
92546 be suitable as input for the *ed* utility, and in the opposite order.

92547 **−r** Apply *diff* recursively to files and directories of the same name when *file1* and *file2*
92548 are both directories.

92549 The *diff* utility shall detect infinite loops; that is, entering a previously visited
92550 directory that is an ancestor of the last file encountered. When it detects an infinite
92551 loop, *diff* shall write a diagnostic message to standard error and shall either recover
92552 its position in the hierarchy or terminate.

92553 **−u** Produce output in a form that provides three lines of unified context.

92554 **−U** *n* Produce output in a form that provides *n* lines of unified context (where *n* shall be
92555 interpreted as a non-negative decimal integer).

92556 **OPERANDS**

92557 The following operands shall be supported:

92558 *file1*, *file2* A pathname of a file to be compared. If either the *file1* or *file2* operand is `'-'`, the
92559 standard input shall be used in its place.

92560 If both *file1* and *file2* are directories, *diff* shall not compare block special files, character special
92561 files, or FIFO special files to any files and shall not compare regular files to directories. Further
92562 details are as specified in Diff Directory Comparison Format (on page 2796). The behavior of *diff*
92563 on other file types is implementation-defined when found in directories.

92564 If only one of *file1* and *file2* is a directory, *diff* shall be applied to the non-directory file and the file
92565 contained in the directory file with a filename that is the same as the last component of the non-
92566 directory file.

92567 **STDIN**

92568 The standard input shall be used only if one of the *file1* or *file2* operands references standard
92569 input. See the INPUT FILES section.

92570 **INPUT FILES**

92571 The input files may be of any type.

92572 **ENVIRONMENT VARIABLES**

92573 The following environment variables shall affect the execution of *diff*:

92574 *LANG* Provide a default value for the internationalization variables that are unset or null.
92575 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
92576 variables used to determine the values of locale categories.)

92577 *LC_ALL* If set to a non-empty string value, override the values of all the other
92578 internationalization variables.

92579 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
92580 characters (for example, single-byte as opposed to multi-byte characters in
92581 arguments and input files).

92582 *LC_MESSAGES*
92583 Determine the locale that should be used to affect the format and contents of
92584 diagnostic messages written to standard error and informative messages written to
92585 standard output.

92586 *LC_TIME* Determine the locale for affecting the format of file timestamps written with the **−C**
92587 and **−c** options.

92588 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

92589 *TZ* Determine the timezone used for calculating file timestamps written with a context
92590 format. If *TZ* is unset or null, an unspecified default timezone shall be used.

92591 **ASYNCHRONOUS EVENTS**

92592 Default.

92593 **STDOUT**

92594 **Diff Directory Comparison Format**

92595 If both *file1* and *file2* are directories, the following output formats shall be used.

92596 In the POSIX locale, each file that is present in only one directory shall be reported using the
92597 following format:

92598 `"Only in %s: %s\n", <directory pathname>, <filename>`

92599 In the POSIX locale, subdirectories that are common to the two directories may be reported with
92600 the following format:

92601 `"Common subdirectories: %s and %s\n", <directory1 pathname>,`
92602 `<directory2 pathname>`

92603 For each file common to the two directories, if the two files are not to be compared: if the two
92604 files have the same device ID and file serial number, or are both block special files that refer to
92605 the same device, or are both character special files that refer to the same device, in the POSIX
92606 locale the output format is unspecified. Otherwise, in the POSIX locale an unspecified format
92607 shall be used that contains the pathnames of the two files.

92608 For each file common to the two directories, if the files are compared and are identical, no

output shall be written. If the two files differ, the following format is written:

`"diff %s %s %s\n", <diff_options>, <filename1>, <filename2>`

where **<***diff_options***>** are the options as specified on the command line.

All directory pathnames listed in this section shall be relative to the original command line arguments. All other names of files listed in this section shall be filenames (pathname components).

Diff Binary Output Format

In the POSIX locale, if one or both of the files being compared are not text files, it is implementation-defined whether *diff* uses the binary file output format or the other formats as specified below. The binary file output format shall contain the pathnames of two files being compared and the string `"differ"`.

If both files being compared are text files, depending on the options specified, one of the following formats shall be used to write the differences.

Diff Default Output Format

The default (without **–e**, **–f**, **–c**, **–C**, **–u**, or **–U** options) *diff* utility output shall contain lines of these forms:

`"%da%d\n", <num1>, <num2>`

`"%da%d,%d\n", <num1>, <num2>, <num3>`

`"%dd%d\n", <num1>, <num2>`

`"%d,%dd%d\n", <num1>, <num2>, <num3>`

`"%dc%d\n", <num1>, <num2>`

`"%d,%dc%d\n", <num1>, <num2>, <num3>`

`"%dc%d,%d\n", <num1>, <num2>, <num3>`

`"%d,%dc%d,%d\n", <num1>, <num2>, <num3>, <num4>`

These lines resemble *ed* subcommands to convert *file1* into *file2*. The line numbers before the action letters shall pertain to *file1*; those after shall pertain to *file2*. Thus, by exchanging *a* for *d* and reading the line in reverse order, one can also determine how to convert *file2* into *file1*. As in *ed*, identical pairs (where *num1*= *num2*) are abbreviated as a single number.

Following each of these lines, *diff* shall write to standard output all lines affected in the first file using the format:

`"<Δ%s", <line>`

and all lines affected in the second file using the format:

`">Δ%s", <line>`

If there are lines affected in both *file1* and *file2* (as with the **c** subcommand), the changes are separated with a line consisting of three **<**hyphen-minus**>** characters:

`"---\n"`

92645 **Diff –e Output Format**

92646 With the –**e** option, a script shall be produced that shall, when provided as input to *ed*, along
92647 with an appended **w** (write) command, convert *file1* into *file2*. Only the **a** (append), **c** (change), **d**
92648 (delete), **i** (insert), and **s** (substitute) commands of *ed* shall be used in this script. Text lines,
92649 except those consisting of the single character <period> (`'.'`), shall be output as they appear in
92650 the file.

92651 **Diff –f Output Format**

92652 With the –**f** option, an alternative format of script shall be produced. It is similar to that
92653 produced by –**e**, with the following differences:

92654 1. It is expressed in reverse sequence; the output of –**e** orders changes from the end of the
92655 file to the beginning; the –**f** from beginning to end.

92656 2. The command form <*lines*> <*command-letter*> used by –**e** is reversed. For example,
92657 10*c* with –**e** would be *c*10 with –**f**.

92658 3. The form used for ranges of line numbers is <space>-separated, rather than
92659 <comma>-separated.

92660 **Diff –c or –C Output Format**

92661 With the –**c** or –**C** option, the output format shall consist of affected lines along with
92662 surrounding lines of context. The affected lines shall show which ones need to be deleted or
92663 changed in *file1*, and those added from *file2*. With the –**c** option, three lines of context, if
92664 available, shall be written before and after the affected lines. With the –**C** option, the user can
92665 specify how many lines of context are written. The exact format follows.

92666 The name and last modification time of each file shall be output in the following format:

92667 `"*** %s %s\n", file1, <file1 timestamp>`
92668 `"--- %s %s\n", file2, <file2 timestamp>`

92669 Each <*file*> field shall be the pathname of the corresponding file being compared. The pathname
92670 written for standard input is unspecified.

92671 In the POSIX locale, each <*timestamp*> field shall be equivalent to the output from the following
92672 command:

92673 `date "+%a %b %e %T %Y"`

92674 without the trailing <newline>, executed at the time of last modification of the corresponding
92675 file (or the current time, if the file is standard input).

92676 Then, the following output formats shall be applied for every set of changes.

92677 First, a line shall be written in the following format:

92678 `"***************\n"`

92679 Next, the range of lines in *file1* shall be written in the following format if the range contains two
92680 or more lines:

92681 `"*** %d,%d ****\n", <beginning line number>, <ending line number>`

92682 and the following format otherwise:

92683 `"*** %d ****\n", <ending line number>`

92684 The ending line number of an empty range shall be the number of the preceding line, or 0 if the

range is at the start of the file.

Next, the affected lines along with lines of context (unaffected lines) shall be written. Unaffected lines shall be written in the following format:

`"ΔΔ%s", <unaffected_line>`

Deleted lines shall be written as:

`"-Δ%s", <deleted_line>`

Changed lines shall be written as:

`"!Δ%s", <changed_line>`

Next, the range of lines in *file2* shall be written in the following format if the range contains two or more lines:

`"--- %d,%d ----\n", <beginning line number>, <ending line number>`

and the following format otherwise:

`"--- %d ----\n", <ending line number>`

Then, lines of context and changed lines shall be written as described in the previous formats. Lines added from *file2* shall be written in the following format:

`"+Δ%s", <added_line>`

Diff –u or –U Output Format

The **–u** or **–U** options behave like the **–c** or **–C** options, except that the context lines are not repeated; instead, the context, deleted, and added lines are shown together, interleaved. The exact format follows.

The name and last modification time of each file shall be output in the following format:

```
"---Δ%s\t%s%sΔ%s\n", file1, <file1 timestamp>, <file1 frac>, <file1 zone>
"+++Δ%s\t%s%sΔ%s\n", file2, <file2 timestamp>, <file2 frac>, <file2 zone>
```

Each <*file*> field shall be the pathname of the corresponding file being compared, or the single character `'-'` if standard input is being compared. However, if the pathname contains a <tab> or a <newline>, or if it does not consist entirely of characters taken from the portable character set, the behavior is implementation-defined.

Each <*timestamp*> field shall be equivalent to the output from the following command:

`date '+%Y-%m-%dΔ%H:%M:%S'`

without the trailing <newline>, executed at the time of last modification of the corresponding file (or the current time, if the file is standard input).

Each <*frac*> field shall be either empty, or a decimal point followed by at least one decimal digit, indicating the fractional-seconds part (if any) of the file timestamp. The number of fractional digits shall be at least the number needed to represent the file's timestamp without loss of information.

Each <*zone*> field shall be of the form `"shhmm"`, where `"shh"` is a signed two-digit decimal number in the range –24 through +25, and `"mm"` is an unsigned two-digit decimal number in the range 00 through 59. It represents the timezone of the timestamp as the number of hours (hh) and minutes (mm) east (+) or west (–) of UTC for the timestamp. If the hours and minutes are both zero, the sign shall be `'+'`. However, if the timezone is not an integral number of minutes

away from UTC, the <*zone*> field is implementation-defined.

Then, the following output formats shall be applied for every set of changes.

First, the range of lines in each file shall be written in the following format:

```
"@@Δ−%sΔ+%sΔ@@", <file1 range>, <file2 range>
```

Each <*range*> field shall be of the form:

```
"%1d", <beginning line number>
```

or:

```
"%1d,1", <beginning line number>
```

if the range contains exactly one line, and:

```
"%1d,%1d", <beginning line number>, <number of lines>
```

otherwise. If a range is empty, its beginning line number shall be the number of the line just before the range, or 0 if the empty range starts the file.

Next, the affected lines along with lines of context shall be written. Each non-empty unaffected line shall be written in the following format:

```
"Δ%s", <unaffected_line>
```

where the contents of the unaffected line shall be taken from *file1*. It is implementation-defined whether an empty unaffected line is written as an empty line or a line containing a single <space> character. This line also represents the same line of *file2*, even though *file2*'s line may contain different contents due to the **−b**. Deleted lines shall be written as:

```
"−%s", <deleted_line>
```

Added lines shall be written as:

```
"+%s", <added_line>
```

The order of lines written shall be the same as that of the corresponding file. A deleted line shall never be written immediately after an added line.

If **−U** *n* is specified, the output shall contain no more than 2*n* consecutive unaffected lines; and if the output contains an affected line and this line is adjacent to up to *n* consecutive unaffected lines in the corresponding file, the output shall contain these unaffected lines. **−u** shall act like **−U**3.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 No differences were found.

1 Differences were found and all differences were successfully output.

>1 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

If lines at the end of a file are changed and other lines are added, *diff* output may show this as a delete and add, as a change, or as a change and add; *diff* is not expected to know which happened and users should not care about the difference in output as long as it clearly shows the differences between the files.

EXAMPLES

If **dir1** is a directory containing a directory named **x**, **dir2** is a directory containing a directory named **x**, **dir1/x** and **dir2/x** both contain files named **date.out**, and **dir2/x** contains a file named **y**, the command:

```
diff -r dir1 dir2
```

could produce output similar to:

```
Common subdirectories: dir1/x and dir2/x
Only in dir2/x: y
diff -r dir1/x/date.out dir2/x/date.out
1c1
< Mon Jul  2 13:12:16 PDT 1990
---
> Tue Jun 19 21:41:39 PDT 1990
```

RATIONALE

The **-h** option was omitted because it was insufficiently specified and does not add to applications portability.

Historical implementations employ algorithms that do not always produce a minimum list of differences; the current language about making every effort is the best this volume of POSIX.1-2024 can do, as there is no metric that could be employed to judge the quality of implementations against any and all file contents. The statement ``This list should be minimal'' clearly implies that implementations are not expected to provide the following output when comparing two 100-line files that differ in only one character on a single line:

```
1,100c1,100
all 100 lines from file1 preceded with "< "
---
all 100 lines from file2 preceded with "> "
```

The ``Only in'' messages required when the **-r** option is specified are not used by most historical implementations if the **-e** option is also specified. It is required here because it provides useful information that must be provided to update a target directory hierarchy to match a source hierarchy. The ``Common subdirectories'' messages are written by System V and 4.3 BSD when the **-r** option is specified. They are allowed here but are not required because they are reporting on something that is the same, not reporting a difference, and are not needed to update a target hierarchy.

The **-c** option, which writes output in a format using lines of context, has been included. The format is useful for a variety of reasons, among them being much improved readability and the ability to understand difference changes when the target file has line numbers that differ from another similar, but slightly different, copy. The *patch* utility is most valuable when working with difference listings using a context format. The BSD version of **-c** takes an optional

92809 argument specifying the amount of context. Rather than overloading −**c** and breaking the Utility
92810 Syntax Guidelines for *diff*, the standard developers decided to add a separate option for
92811 specifying a context diff with a specified amount of context (−**C**). Also, the format for context
92812 *diff*s was extended slightly in 4.3 BSD to allow multiple changes that are within context lines
92813 from each other to be merged together. The output format contains an additional four <asterisk>
92814 characters after the range of affected lines in the first filename. This was to provide a flag for old
92815 programs (like old versions of *patch*) that only understand the old context format. The version of
92816 context described here does not require that multiple changes within context lines be merged,
92817 but it does not prohibit it either. The extension is upwards-compatible, so any vendors that wish
92818 to retain the old version of *diff* can do so by adding the extra four <asterisk> characters (that is,
92819 utilities that currently use *diff* and understand the new merged format will also understand the
92820 old unmerged format, but not *vice versa*).

92821 The −**u** and −**U** options of GNU *diff* have been included. Their output format, designed by
92822 Wayne Davison, takes up less space than −**c** and −**C** format, and in many cases is easier to read.
92823 The format's timestamps do not vary by locale, so *LC_TIME* does not affect it. The format's line
92824 numbers are rendered with the `%1d` format, not `%d`, because the file format notation rules would
92825 allow extra <blank> characters to appear around the numbers.

92826 The substitute command was added as an additional format for the −**e** option. This was added
92827 to provide implementations with a way to fix the classic ``dot alone on a line'' bug present in
92828 many versions of *diff*. Since many implementations have fixed this bug, the standard developers
92829 decided not to standardize broken behavior, but rather to provide the necessary tool for fixing
92830 the bug. One way to fix this bug is to output two periods whenever a lone period is needed, then
92831 terminate the append command with a period, and then use the substitute command to convert
92832 the two periods into one period.

92833 The BSD-derived −**r** option was added to provide a mechanism for using *diff* to compare two file
92834 system trees. This behavior is useful, is standard practice on all BSD-derived systems, and is not
92835 easily reproducible with the *find* utility.

92836 The requirement that *diff* not compare files in some circumstances, even though they have the
92837 same name, is based on the actual output of historical implementations. The specified behavior
92838 precludes the problems arising from running into FIFOs and other files that would cause *diff* to
92839 hang waiting for input with no indication to the user that *diff* was hung. An earlier version of
92840 this standard specified the output format more precisely, but in practice this requirement was
92841 widely ignored and the benefit of standardization seemed small, so it is now unspecified. In
92842 most common usage, *diff* −**r** should indicate differences in the file hierarchies, not the difference
92843 of contents of devices pointed to by the hierarchies.

92844 Many early implementations of *diff* require seekable files. Since the System Interfaces volume of
92845 POSIX.1-2024 supports named pipes, the standard developers decided that such a restriction
92846 was unreasonable. Note also that the allowed filename – almost always refers to a pipe.

92847 No directory search order is specified for *diff*. The historical ordering is, in fact, not optimal, in
92848 that it prints out all of the differences at the current level, including the statements about all
92849 common subdirectories before recursing into those subdirectories.

92850 The message:

92851 `"diff %s %s %s\n", <diff_options>, <filename1>, <filename2>`

92852 does not vary by locale because it is the representation of a command, not an English sentence.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

cmp, *comm*, *ed*, *find*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **–f** option is added.

The output format for **–c** or **–C** format is changed to align with changes to the IEEE P1003.2b draft standard resulting from IEEE PASC Interpretation 1003.2 #71.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/20 is applied, changing the STDOUT section. This changes the specification of *diff* **–c** so that it agrees with existing practice when contexts contain zero lines or one line.

Issue 7

Austin Group Interpretations 1003.1-2001 #115 and #114 are applied.

Austin Group Interpretation 1003.1-2001 #192 is applied, clarifying the behavior if both files are non-text files.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

SD5-XCU-ERN-103 and SD5-XCU-ERN-120 are applied, adding the **–u** option.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0082 [584], XCU/TC2-2008/0083 [950], XCU/TC2-2008/0084 [969], and XCU/TC2-2008/0085 [929] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1498 is applied, changing the EXIT STATUS section.

92890 **NAME**

92891 dirname — return the directory portion of a pathname

92892 **SYNOPSIS**

92893 `dirname` *`string`*

92894 **DESCRIPTION**

92895 The *string* operand shall be treated as a pathname, as defined in XBD Section 3.254 (on page 68),
92896 and shall be converted to a pathname of the directory containing the entry of the final pathname
92897 component. The resulting string shall be written to standard output. The *dirname* utility shall not
92898 perform pathname resolution; the result shall not be affected by whether or not a file with the
92899 pathname *string* exists or by its file type. Trailing `'/'` characters in *string* that are not also
92900 leading `'/'` characters shall not be counted as part of the pathname. If the pathname does not
92901 contain a `'/'`, the resulting string shall be `"."`. If *string* is an empty string, the resulting string
92902 shall be `"."`.

92903 It is unspecified whether redundant `'/'` characters and `'.'` pathname components in *string* are
92904 removed after determining the pathname to output. However, `".."` pathname components
92905 occurring prior to the final component shall not be removed.

92906 **OPTIONS**

92907 None.

92908 **OPERANDS**

92909 The following operand shall be supported:

92910 *string* A string.

92911 **STDIN**

92912 Not used.

92913 **INPUT FILES**

92914 None.

92915 **ENVIRONMENT VARIABLES**

92916 The following environment variables shall affect the execution of *dirname*:

92917 *LANG* Provide a default value for the internationalization variables that are unset or null.
92918 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
92919 variables used to determine the values of locale categories.)

92920 *LC_ALL* If set to a non-empty string value, override the values of all the other
92921 internationalization variables.

92922 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
92923 characters (for example, single-byte as opposed to multi-byte characters in
92924 arguments).

92925 *LC_MESSAGES*
92926 Determine the locale that should be used to affect the format and contents of
92927 diagnostic messages written to standard error.

92928 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

92929 **ASYNCHRONOUS EVENTS**

92930 Default.

92931 **STDOUT**

92932 The *dirname* utility shall write a line to the standard output in the following format:

92933 `"%s\n", <resulting string>`

92934 **STDERR**

92935 The standard error shall be used only for diagnostic messages.

92936 **OUTPUT FILES**

92937 None.

92938 **EXTENDED DESCRIPTION**

92939 None.

92940 **EXIT STATUS**

92941 The following exit values shall be returned:

92942 0 Successful completion.

92943 >0 An error occurred.

92944 **CONSEQUENCES OF ERRORS**

92945 Default.

92946 **APPLICATION USAGE**

92947 The definition of *pathname* specifies implementation-defined behavior for pathnames starting
92948 with two <slash> characters. Therefore, applications shall not arbitrarily add <slash> characters
92949 to the beginning of a pathname unless they can ensure that there are more or less than two or are
92950 prepared to deal with the implementation-defined consequences.

92951 **EXAMPLES**

92952 The EXAMPLES section of the *basename*( ) function (see XSH *basename( )*) includes a table
92953 showing examples of the results of processing several sample pathnames by the *basename*( ) and
92954 *dirname*( ) functions and by the *basename* and *dirname* utilities.

92955 See also the examples for the *basename* utility.

92956 **RATIONALE**

92957 The behaviors of *basename* and *dirname* in this volume of POSIX.1-2024 have been coordinated so
92958 that when *string* is a valid pathname:

92959 `$(basename -- "string")`

92960 would be a valid filename for the file in the directory:

92961 `$(dirname -- "string")`

92962 This would not work for the versions of these utilities in early proposals due to the way
92963 processing of trailing <slash> characters was specified. Consideration was given to leaving
92964 processing unspecified if there were trailing <slash> characters, but this cannot be done; XBD
92965 Section 3.254 (on page 68) allows trailing <slash> characters. The *basename* and *dirname* utilities
92966 have to specify consistent handling for all valid pathnames.

92967 The *dirname* utility is not specified in terms of the *dirname*( ) function, because the two may
92968 produce slightly different output where both output forms are still compliant. An
92969 implementation should prefer the shortest output possible; however, this is not required, in part
92970 because earlier versions of the standard did not permit elision of redundant <slash> characters
92971 or dot (`'.'`) components. Removal of the dot-dot (`".."`) pathname component is not permitted,
92972 because eliding it correctly would require performing pathname resolution to ensure the
92973 resulting string would still point to the correct pathname if the original string resolved as a

pathname. On implementations where pathname "//" has an implementation-defined meaning distinct from the pathname "/", the dirname of "//" will be "//".

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

Section 2.5 (on page 2478), *basename*

XBD Section 3.254 (on page 68), Chapter 8 (on page 167)

XSH *basename*( ), *dirname*( )

CHANGE HISTORY

First released in Issue 2.

Issue 7

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0083 [192,430], XCU/TC1-2008/0084 [192], and XCU/TC1-2008/0085 [192] are applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0086 [612], XCU/TC2-2008/0087 [620], and XCU/TC2-2008/0088 [612] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1073 is applied, replacing the DESCRIPTION section with one that matches the *dirname*( ) function.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

du — estimate file space usage

SYNOPSIS

```
du [-a|-s] [-kx] [-H|-L] [file...]
```

DESCRIPTION

By default, the *du* utility shall write to standard output the size of the file space allocated to, and the size of the file space allocated to each subdirectory of, the file hierarchy rooted in each of the specified files. By default, when a symbolic link is encountered on the command line or in the file hierarchy, *du* shall count the size of the symbolic link (rather than the file referenced by the link), and shall not follow the link to another portion of the file hierarchy. The size of the file space allocated to a file of type directory shall be defined as the sum total of space allocated to all files in the file hierarchy rooted in the directory plus the space allocated to the directory itself.

When *du* cannot *stat*( ) files or *stat*( ) or read directories, it shall report an error condition and the final exit status is affected. A file that occurs multiple times shall be counted and written for only one entry, even if the occurrences are under different file operands. The directory entry that is selected in the report is unspecified. By default, file sizes shall be written in 512-byte units, rounded up to the next 512-byte unit.

OPTIONS

The *du* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–a In addition to the default output, report the size of each file not of type directory in the file hierarchy rooted in the specified file. The **–a** option shall not affect whether non-directories given as *file* operands are listed.

–H If a symbolic link is specified on the command line, *du* shall count the size of the file or file hierarchy referenced by the link.

–k Write the files sizes in units of 1 024 bytes, rather than the default 512-byte units.

–L If a symbolic link is specified on the command line or encountered during the traversal of a file hierarchy, *du* shall count the size of the file or file hierarchy referenced by the link.

–s Instead of the default output, report only the total sum for each of the specified files.

–x When evaluating file sizes, evaluate only those files that have the same device as the file specified by the *file* operand.

Specifying more than one of the mutually-exclusive options **–H** and **–L** shall not be considered an error. The last option specified shall determine the behavior of the utility.

OPERANDS

The following operand shall be supported:

file The pathname of a file whose size is to be written. If no *file* is specified, the current directory shall be used.

STDIN

Not used.

93041 **INPUT FILES**

93042 None.

93043 **ENVIRONMENT VARIABLES**

93044 The following environment variables shall affect the execution of *du*:

93045 *LANG* Provide a default value for the internationalization variables that are unset or null.
93046 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
93047 variables used to determine the values of locale categories.)

93048 *LC_ALL* If set to a non-empty string value, override the values of all the other
93049 internationalization variables.

93050 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
93051 characters (for example, single-byte as opposed to multi-byte characters in
93052 arguments).

93053 *LC_MESSAGES*
93054 Determine the locale that should be used to affect the format and contents of
93055 diagnostic messages written to standard error.

93056 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

93057 **ASYNCHRONOUS EVENTS**

93058 Default.

93059 **STDOUT**

93060 The output from *du* shall consist of the amount of space allocated to a file and the name of the
93061 file, in the following format:

93062 `"%d %s\n", <size>, <pathname>`

93063 **STDERR**

93064 The standard error shall be used only for diagnostic messages.

93065 **OUTPUT FILES**

93066 None.

93067 **EXTENDED DESCRIPTION**

93068 None.

93069 **EXIT STATUS**

93070 The following exit values shall be returned:

93071 0 Successful completion.

93072 >0 An error occurred.

93073 **CONSEQUENCES OF ERRORS**

93074 Default.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

The use of 512-byte units is historical practice and maintains compatibility with *ls* and other utilities in this volume of POSIX.1-2024. This does not mandate that the file system itself be based on 512-byte blocks. The **−k** option was added as a compromise measure. It was agreed by the standard developers that 512 bytes was the best default unit because of its complete historical consistency on System V (*versus* the mixed 512/1 024-byte usage on BSD systems), and that a **−k** option to switch to 1 024-byte units was a good compromise. Users who prefer the 1 024-byte quantity can easily alias *du* to *du* **−k** without breaking the many historical scripts relying on the 512-byte units.

The **−b** option was added to an early proposal to provide a resolution to the situation where System V and BSD systems give figures for file sizes in *blocks*, which is an implementation-defined concept. (In common usage, the block size is 512 bytes for System V and 1 024 bytes for BSD systems.) However, **−b** was later deleted, since the default was eventually decided as 512-byte units.

Historical file systems provided no way to obtain exact figures for the space allocation given to files. There are two known areas of inaccuracies in historical file systems: cases of *indirect blocks* being used by the file system or *sparse* files yielding incorrectly high values. An indirect block is space used by the file system in the storage of the file, but that need not be counted in the space allocated to the file. A *sparse* file is one in which an *lseek*( ) call has been made to a position beyond the end of the file and data has subsequently been written at that point. A file system need not allocate all the intervening zero-filled blocks to such a file. It is up to the implementation to define exactly how accurate its methods are.

The **−a** and **−s** options were mutually-exclusive in the original version of *du*. The POSIX Shell and Utilities description is implied by the language in the SVID where **−s** is described as causing ``only the grand total'' to be reported. Some systems may produce output for **−sa**, but a Strictly Conforming POSIX Shell and Utilities Application cannot use that combination.

The **−a** and **−s** options were adopted from the SVID except that the System V behavior of not listing non-directories explicitly given as operands, unless the **−a** option is specified, was considered a bug; the BSD-based behavior (report for all operands) is mandated. The default behavior of *du* in the SVID with regard to reporting the failure to read files (it produces no messages) was considered counter-intuitive, and thus it was specified that the POSIX Shell and Utilities default behavior shall be to produce such messages. These messages can be turned off with shell redirection to achieve the System V behavior.

The **−x** option is historical practice on recent BSD systems. It has been adopted by this volume of POSIX.1-2024 because there was no other historical method of limiting the *du* search to a single file hierarchy. This limitation of the search is necessary to make it possible to obtain file space usage information about a file system on which other file systems are mounted, without having to resort to a lengthy *find* and *awk* script.

The use of the **−L** option, or of multiple *file* operands, requires that *du* track all file entries encountered, even with a link count of one. However, when **−L** is not used and only a single *file* operand is given, an implementation can optimize by only tracking files with a link count greater than one, since in that scenario, those are the only files that could be encountered more than once.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

ls

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *fstatat*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The APPLICATION USAGE section is added.

The obsolescent **–r** option is removed.

The Open Group Corrigendum U025/3 is applied. The *du* utility is reinstated, as it had incorrectly been marked LEGACY in Issue 5.

The **–H** and **–L** options for symbolic links are added as described in the IEEE P1003.2b draft standard.

Issue 7

The *du* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0089 [527] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 539 is applied, requiring a file that occurs multiple times to be counted and written for only one entry.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

echo — write arguments to standard output

SYNOPSIS

```
echo [string...]
```

DESCRIPTION

The *echo* utility writes its arguments to standard output, followed by a <newline>. If there are no arguments, only the <newline> is written.

OPTIONS

The *echo* utility shall not recognize the `"--"` argument in the manner specified by Guideline 10 of XBD Section 12.2 (on page 215); `"--"` shall be recognized as a string operand.

Implementations shall not support any options.

OPERANDS

The following operands shall be supported:

string A string to be written to standard output. If the first operand consists of a `'-'` followed by one or more characters from the set {`'e'`, `'E'`, `'n'`}, or if any of the operands contain a <backslash> character, the results are implementation-defined.

XSI On XSI-conformant systems, if the first operand consists of a `'-'` followed by one or more characters from the set {`'e'`, `'E'`, `'n'`}, it shall be treated as a string to be written. The following character sequences shall be recognized on XSI-conformant systems within any of the arguments:

`\a` Write an <alert>.

`\b` Write a <backspace>.

`\c` Suppress the <newline> that otherwise follows the final argument in the output. All characters following the `'\c'` in the arguments shall be ignored.

`\f` Write a <form-feed>.

`\n` Write a <newline>.

`\r` Write a <carriage-return>.

`\t` Write a <tab>.

`\v` Write a <vertical-tab>.

`\\` Write a <backslash> character.

`\0`*num* Write an 8-bit value that is the zero, one, two, or three-digit octal number *num*.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *echo*:

93193 *LANG* Provide a default value for the internationalization variables that are unset or null.
93194 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
93195 variables used to determine the values of locale categories.)

93196 *LC_ALL* If set to a non-empty string value, override the values of all the other
93197 internationalization variables.

93198 XSI *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
93199 characters (for example, single-byte as opposed to multi-byte characters in
93200 arguments).

93201 *LC_MESSAGES*
93202 Determine the locale that should be used to affect the format and contents of
93203 diagnostic messages written to standard error.

93204 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

93205 **ASYNCHRONOUS EVENTS**
93206 Default.

93207 **STDOUT**
93208 The *echo* utility arguments shall be separated by single <space> characters and a <newline>
93209 XSI character shall follow the last argument. Output transformations shall occur based on the
93210 escape sequences in the input. See the OPERANDS section.

93211 **STDERR**
93212 The standard error shall be used only for diagnostic messages.

93213 **OUTPUT FILES**
93214 None.

93215 **EXTENDED DESCRIPTION**
93216 None.

93217 **EXIT STATUS**
93218 The following exit values shall be returned:

93219 0 Successful completion.

93220 >0 An error occurred.

93221 **CONSEQUENCES OF ERRORS**
93222 Default.

93223 **APPLICATION USAGE**
93224 It is not possible to use *echo* portably across all POSIX systems unless escape sequences are
93225 omitted, and the first argument does not consist of a `'-'` followed by one or more characters
93226 from the set {`'e'`,`'E'`,`'n'`}.

93227 The *printf* utility can be used portably to emulate any of the traditional behaviors of the *echo*
93228 utility as follows (assuming that *IFS* has its standard value or is unset):

93229 • The historic System V *echo* and the requirements on XSI implementations in this volume of
93230 POSIX.1-2024 are equivalent to:

93231 ```
printf "%b\n" "$*"
```

93232 • The BSD *echo* is equivalent to:

93233 ```
if [ "X$1" = "X-n" ]
93234 then
```



```
            shift
            printf "%s" "$*"
        else
            printf "%s\n" "$*"
        fi
```

New applications are encouraged to use *printf* instead of *echo*.

EXAMPLES

None.

RATIONALE

The *echo* utility has not been made obsolescent because of its extremely widespread use in historical applications. Conforming applications that wish to do prompting without <newline> characters or that could possibly be expecting to echo a string consisting of a `'-'` followed by one or more characters from the set {`'e'`, `'E'`, `'n'`} should use the *printf* utility.

At the time that the IEEE Std 1003.2-1992 standard was being developed, the two different historical versions of *echo* that were considered for standardization varied in incompatible ways.

The BSD *echo* checked the first argument for the string **–n** which caused it to suppress the <newline> that would otherwise follow the final argument in the output.

The System V *echo* treated all arguments as strings to be written, but allowed escape sequences within them, as described for XSI implementations in the OPERANDS section, including `\c` to suppress a trailing <newline>.

Thus the IEEE Std 1003.2-1992 standard said that the behavior was implementation-defined if the first operand is **–n** or if any of the operands contain a <backslash> character. It also specified that the *echo* utility does not support Utility Syntax Guideline 10 because historical applications depended on *echo* to echo *all* of its arguments, except for the **–n** first argument in the BSD version.

The Single UNIX Specification, Version 1 required the System V behavior, and this became the XSI requirement when Version 2 and POSIX.2 were merged with POSIX.1 to form the joint IEEE Std 1003.1-2001 / Single UNIX Specification, Version 3 standard.

This standard now treats a first operand of **–e** or **–E** the same as **–n** in recognition that support for them has become more widespread in non-XSI implementations. Where supported, **–e** enables processing of escape sequences in the remaining operands (in situations where it is disabled by default), and **–E** disables it (in situations where it is enabled by default). A first operand containing a combination of these three letters, in the same manner as option grouping, also results in implementation-defined behavior.

FUTURE DIRECTIONS

None.

SEE ALSO

printf

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 5

In the OPTIONS section, the last sentence is changed to indicate that implementations ``do not'' support any options; in the previous issue this said ``need not''.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- A set of character sequences is defined as *string* operands.
- *LC_CTYPE* is added to the list of environment variables affecting *echo*.
- In the OPTIONS section, implementations shall not support any options.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/21 is applied, so that the *echo* utility can accommodate historical BSD behavior.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1222 is applied, making the results implementation-defined, on systems that are not XSI-conformant, if the first operand consists of a `'-'` followed by one or more characters from the set {`'e'`, `'E'`, `'n'`}.

NAME

ed — edit text

SYNOPSIS

```
ed [-p string] [-s] [file]
```

DESCRIPTION

The *ed* utility is a line-oriented text editor that uses two modes: *command mode* and *input mode*. In command mode the input characters shall be interpreted as commands, and in input mode they shall be interpreted as text. See the EXTENDED DESCRIPTION section.

If an operand is `'-'`, the results are unspecified.

OPTIONS

The *ed* utility shall conform to XBD Section 12.2 (on page 215), except for the unspecified usage of `'-'`.

The following options shall be supported:

-p *string* Use *string* as the prompt string when in command mode. By default, there shall be no prompt string.

-s Suppress the writing of byte counts by **e**, **E**, **r**, and **w** commands and of the `'!'` prompt after a !*command*.

OPERANDS

The following operand shall be supported:

file If the *file* argument is given, *ed* shall perform the effect of an **e** command on the pathname *file* before accepting commands from the standard input, except that *file* can contain a <newline>, even though this is not possible for the argument to the **e** command.

STDIN

The standard input shall be a text file consisting of commands, as described in the EXTENDED DESCRIPTION section.

INPUT FILES

The input files shall be text files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *ed*:

HOME Determine the pathname of the user's home directory.

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes within regular expressions.

93337 *LC_MESSAGES*
93338 Determine the locale that should be used to affect the format and contents of
93339 diagnostic messages written to standard error and informative messages written to
93340 standard output.

93341 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

93342 **ASYNCHRONOUS EVENTS**
93343 The *ed* utility shall take the standard action for all signals (see the ASYNCHRONOUS EVENTS
93344 section in Section 1.4, on page 2462) with the following exceptions:

93345 SIGINT The *ed* utility shall interrupt its current activity, write the string `"?\n"` to standard
93346 output, and return to command mode (see the EXTENDED DESCRIPTION
93347 section).

93348 SIGHUP If the buffer is not empty and the buffer change flag is currently set to either
93349 **changed** or **changed-and-warned** (see the EXTENDED DESCRIPTION section),
93350 the *ed* utility shall attempt to write a copy of the buffer in a file. First, the file
93351 named **ed.hup** in the current directory shall be used; if that fails, the file named
93352 **ed.hup** in the directory named by the *HOME* environment variable shall be used.
93353 In any case, the *ed* utility shall exit without writing the file to the currently
93354 remembered pathname and without returning to command mode.

93355 SIGQUIT The *ed* utility shall ignore this event.

93356 **STDOUT**
93357 Various editing commands and the prompting feature (see **−p**) write to standard output, as
93358 described in the EXTENDED DESCRIPTION section.

93359 **STDERR**
93360 The standard error shall be used for diagnostic messages and may be used for warning
93361 messages.

93362 **OUTPUT FILES**
93363 The output files shall be text files whose formats are dependent on the editing commands given.

93364 **EXTENDED DESCRIPTION**
93365 The *ed* utility shall operate on a copy of the file it is editing; changes made to the copy shall have
93366 no effect on the file until a **w** (write) command is given. The copy of the text is called the *buffer*.
93367 The *ed* utility shall keep track of whether the buffer has been modified. This shall be maintained
93368 as if via a tri-state internal flag with the state values **unchanged**, **changed**, and **changed-and-**
93369 **warned**, which is:

93370 • Initially set to **unchanged**

93371 • Set to **changed** by any command that modifies the buffer

93372 • Set to **unchanged** by an **e** or **E** command that reloads (or empties) the buffer, or a **w**
93373 command that writes the entire buffer

93374 • Set to either **changed-and-warned** or **unchanged** by an **e** or **q** command that warns an
93375 attempt was made to destroy the editor buffer

93376 A command that makes changes to the buffer in such a way that its contents are the same after
93377 the command (for example **s/a/a/**) shall be considered to have modified the buffer, unless
93378 explicitly stated otherwise. In the remainder of the description, this flag is referred to as the
93379 *buffer change flag*.

93380 Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a

single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses very often can be omitted. If the **−p** option is specified, the prompt string shall be written to standard output before each command is read.

In general, only one command can appear on a line. Certain commands allow text to be input. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, no commands shall be recognized; all input is merely collected. Input mode is terminated by entering a line consisting of two characters: a <period> (`'.'`) followed by a <newline>. This line is not considered part of the input text.

Regular Expressions in ed

The *ed* utility shall support basic regular expressions, as described in XBD Section 9.3 (on page 181). Since regular expressions in *ed* are always matched against single lines (excluding the terminating <newline> characters), never against any larger section of text, there is no way for a regular expression to match a <newline>.

A null RE shall be equivalent to the last RE encountered.

Regular expressions are used in addresses to specify lines, and in some commands (for example, the **s** substitute command) to specify portions of a line to be substituted.

The start and end of a regular expression (RE) are marked by a delimiter character (although in some circumstances the end delimiter can be omitted). In addresses, the delimiter is either <slash> or <question-mark>. In commands, other characters can be used as the delimiter, as specified in the description of the command. Within the RE (as an *ed* extension to the BRE syntax), the delimiter shall not terminate the RE if it is the second character of an escape sequence (see XBD Section 9.1, on page 179) and the escaped delimiter shall be treated as that literal character in the RE (losing any special meaning it would have had if it was not used as the delimiter and was not escaped). In addition, the delimiter character shall not terminate the RE when it appears within a bracket expression, and shall have its normal meaning in the bracket expression. For example, the command `"g%[%]%p"` is equivalent to `"g/[%]/p"`, and the command `"s-[0-9]--g"` is equivalent to `"s/[0-9]//g"`.

Addresses in ed

Addressing in *ed* relates to the current line. Generally, the current line is the last line affected by a command. The current line number is the address of the current line. If the edit buffer is not empty, the initial value for the current line shall be the last line in the edit buffer; otherwise, zero.

Addresses shall be constructed as follows:

1. The <period> character (`'.'`) shall address the current line.
2. The <dollar-sign> character (`'$'`) shall address the last line of the edit buffer.
3. The positive decimal number *n* shall address the *n*th line of the edit buffer.
4. The <apostrophe>-x character pair (`"'x"`) shall address the line marked with the mark name character *x*, which shall be a lowercase letter from the portable character set. It shall be an error if the character has not been set to mark a line or if the line that was marked is not currently present in the edit buffer.
5. A BRE enclosed by <slash> characters (`'/'`) shall address the first line found by searching forwards from the line following the current line toward the end of the edit buffer and stopping at the first line for which the line excluding the terminating <newline> matches the BRE. The BRE consisting of a null BRE delimited by a pair of

93425 <slash> characters shall address the next line for which the line excluding the terminating
93426 <newline> matches the last BRE encountered. In addition, the second <slash> can be
93427 omitted at the end of a command line. Within the BRE, a <backslash>-<slash> pair (`"\/"`)
93428 shall represent a literal <slash> instead of the BRE delimiter. If necessary, the search shall
93429 wrap around to the beginning of the buffer and continue up to and including the current
93430 line, so that the entire buffer is searched.

93431 6. A BRE enclosed by <question-mark> characters (`'?'`) shall address the first line found by
93432 searching backwards from the line preceding the current line toward the beginning of the
93433 edit buffer and stopping at the first line for which the line excluding the terminating
93434 <newline> matches the BRE. The BRE consisting of a null BRE delimited by a pair of
93435 <question-mark> characters (`"??"`) shall address the previous line for which the line
93436 excluding the terminating <newline> matches the last BRE encountered. In addition, the
93437 second <question-mark> can be omitted at the end of a command line. Within the BRE, a
93438 <backslash>-<question-mark> pair (`"\?"`) shall represent a literal <question-mark>
93439 instead of the BRE delimiter. If necessary, the search shall wrap around to the end of the
93440 buffer and continue up to and including the current line, so that the entire buffer is
93441 searched.

93442 7. A <plus-sign> (`'+'`) or <hyphen-minus> character (`'-'`) followed by a decimal number
93443 shall address the current line plus or minus the number. A <plus-sign> or <hyphen-
93444 minus> character not followed by a decimal number shall address the current line plus or
93445 minus 1.

93446 Addresses can be followed by zero or more address offsets, optionally <blank>-separated.
93447 Address offsets are constructed as follows:

93448 • A <plus-sign> or <hyphen-minus> character followed by a decimal number shall add or
93449 subtract, respectively, the indicated number of lines to or from the address. A <plus-sign>
93450 or <hyphen-minus> character not followed by a decimal number shall add or subtract 1 to
93451 or from the address.

93452 • A decimal number shall add the indicated number of lines to the address.

93453 It shall not be an error for an intermediate address value to be less than zero or greater than the
93454 last line in the edit buffer. It shall be an error for the final address value to be less than zero or
93455 greater than the last line in the edit buffer. It shall be an error if a search for a BRE fails to find a
93456 matching line.

93457 Commands accept zero, one, or two addresses. If one or more addresses are provided to a
93458 command that requires zero addresses, it shall be an error. Otherwise, if more than the
93459 maximum number of accepted addresses are provided to a command, the addresses shall be
93460 evaluated from first to last and then discarded, until the maximum number of accepted
93461 addresses for that command remain.

93462 Addresses shall be separated from each other by a <comma> (`','`) or <semicolon> character
93463 (`';'`). In the case of a <semicolon> separator, the current line (`'.'`) shall be set to the first
93464 address, and only then shall the second address be calculated. This feature can be used to
93465 determine the starting line for forwards and backwards searches; see rules 5. and 6.

93466 Addresses can be omitted on either side of the <comma> or <semicolon> separator, in which
93467 case the resulting address pairs shall be as follows:

Specified	Resulting
`,`	`1 , $`
`, addr`	`1 , addr`
`addr ,`	`addr , addr`
`;`	`. ; $`
`; addr`	`. ; addr`
`addr ;`	`addr ; addr`

If an address is omitted between two separators, the rule shall be applied to the first separator and the resulting second address shall be used as the first address for the second separator. For example, with the address list `",,"` the first `','` becomes `"1,$"` and the `'$'` is treated as the first address for the second `','`, resulting in `"1,$,$"`.

Any <blank> characters included between addresses, address separators, or address offsets shall be ignored.

Commands in ed

In the following list of *ed* commands, the default addresses are shown in parentheses. The number of addresses shown in the default shall be the number expected by the command. The parentheses are not part of the address; they show that the given addresses are the default.

It is generally invalid for more than one command to appear on a line. However, any command (except **e**, **E**, **f**, **q**, **Q**, **r**, **w**, and **!**) can be suffixed by the letter **l**, **n**, or **p**; in which case, except for the **l**, **n**, and **p** commands, the command shall be executed and then the new current line shall be written as described below under the **l**, **n**, and **p** commands. When an **l**, **n**, or **p** suffix is used with an **l**, **n**, or **p** command, the command shall write to standard output as described below, but it is unspecified whether the suffix writes the current line again in the requested format or whether the suffix has no effect. For example, the **pl** command (base **p** command with an **l** suffix) shall either write just the current line or write it twice—once as specified for **p** and once as specified for **l**. Also, the **g**, **G**, **v**, and **V** commands shall take a command as a parameter.

Each address component can be preceded by zero or more <blank> characters. The command letter can be preceded by zero or more <blank> characters. If a suffix letter (**l**, **n**, or **p**) is given, the application shall ensure that it immediately follows the command.

The **e**, **E**, **f**, **r**, and **w** commands shall take an optional *file* parameter, separated from the command letter by one or more <blank> characters.

If the buffer change flag is currently set to **changed**, *ed* shall warn the user if an attempt is made to destroy the editor buffer via the **e** or **q** commands. The *ed* utility shall write the string:

`"?\n"`

(followed by an explanatory message if *help mode* has been enabled via the **H** command) to standard output and shall continue in command mode with the buffer change flag set to either **changed-and-warned** or **unchanged** and the current line number unchanged. If another **e** or **q** command is then attempted with no intervening command that sets the buffer change flag to **changed**, it shall take effect.

If a terminal disconnect (see XBD Chapter 11 (on page 199), Modem Disconnect and Closing a Device Terminal), is detected:

- If accompanied by a SIGHUP signal, the *ed* utility shall operate as described in the ASYNCHRONOUS EVENTS section for a SIGHUP signal.

- If not accompanied by a SIGHUP signal, the *ed* utility shall act as if an end-of-file had been detected on standard input.

If an end-of-file is detected on standard input:

- If the *ed* utility is in input mode, *ed* shall terminate input mode and return to command mode. It is unspecified if any partially entered lines (that is, input text without a terminating <newline>) are discarded from the input text.
- If the *ed* utility is in command mode, it shall act as if a **q** command had been entered.

In the following commands, if a closing delimiter would be the last character before a <newline> then that character can be omitted with the behavior shown:

- For the **g** and **v** commands the addressed line(s) shall be written as if a closing delimiter followed by a **p** were appended to the command.
- For the **G** and **V** commands no additional action shall be taken.
- For the **s** command, only the closing delimiter of the replacement can be omitted, in which case the result of the substitution shall be written as if the **p** flag were appended.
- For the null command, the addressed line(s) shall be written as if the closing delimiter were appended.

For example, the following pairs of commands are equivalent:

```
s/s1/s2    s/s1/s2/p
g/s1       g/s1/p
?s1        ?s1?
```

If an invalid command is entered, *ed* shall write the string:

```
"?\n"
```

(followed by an explanatory message if *help mode* has been enabled via the **H** command) to standard output and shall continue in command mode with the current line number unchanged.

Append Command

Synopsis:

```
(.)a
<text>
.
```

The **a** command shall read the given text and append it after the addressed line; the current line number shall become the address of the last inserted line or, if there were none, the addressed line. Address 0 shall be valid for this command; it shall cause the appended text to be placed at the beginning of the buffer. If <*text*> is empty (that is, the terminating `'.'` immediately follows the `'a'`), the buffer change flag shall not be altered.

Change Command

Synopsis:

```
(.,.)c
<text>
.
```

The **c** command shall delete the addressed lines, then accept input text that replaces these lines; the current line shall be set to the address of the last line input; or, if there were none, at the line after the last line deleted; if the lines deleted were originally at the end of the buffer, the current line number shall be set to the address of the new last line; if no lines remain in the buffer, the

93552 current line number shall be set to zero.

93553 **Delete Command**

93554 *Synopsis*: `(.,.)d`

93555 The **d** command shall delete the addressed lines from the buffer. The address of the line after the
93556 last line deleted shall become the current line number; if the lines deleted were originally at the
93557 end of the buffer, the current line number shall be set to the address of the new last line; if no
93558 lines remain in the buffer, the current line number shall be set to zero.

93559 **Edit Command**

93560 *Synopsis*: `e [`*`file`*`]`

93561 The **e** command shall delete the entire contents of the buffer and then read in the file named by
93562 the pathname *file*. The current line number shall be set to the address of the last line of the
93563 buffer. If no pathname is given, the currently remembered pathname, if any, shall be used (see
93564 the **f** command). If the pathname names a file that does not exist and the buffer change flag is
93565 currently set to **unchanged**, it is unspecified whether this is treated as an error, or whether the
93566 resulting buffer is emptied and a warning is written to standard error instead of writing the byte
93567 count to standard out. The number of bytes read shall be written to standard output, unless the
93568 –**s** option was specified, in the following format:

93569 `"%d\n", <`*`number of bytes read`*`>`

93570 The name *file* shall be remembered for possible use as a default pathname in subsequent **e**, **E**, **r**,
93571 and **w** commands. If *file* is replaced by `'!'`, the rest of the line shall be taken to be a shell
93572 command line whose output is to be read. Such a shell command line shall not be remembered
93573 as the current *file*. All marks shall be discarded upon the completion of a successful **e** command.
93574 If the buffer change flag is currently set to **changed**, the user shall be warned, as described
93575 previously.

93576 **Edit Without Checking Command**

93577 *Synopsis*: `E [`*`file`*`]`

93578 The **E** command shall possess all properties and restrictions of the **e** command except that the
93579 editor shall not check the current state of the buffer change flag.

93580 **Filename Command**

93581 *Synopsis*: `f [`*`file`*`]`

93582 If *file* is given, the **f** command shall change the currently remembered pathname to *file*, whether
93583 or not *file* names an existing file; whether the name is changed or not, it shall then write the
93584 (possibly new) currently remembered pathname to the standard output in the following format:

93585 `"%s\n", <`*`pathname`*`>`

93586 The current line number shall be unchanged.

93587 **Global Command**

93588 *Synopsis*: `(1,$)g/`*RE*`/`*command list*

93589 In the **g** command, the first step shall be to mark every line for which the line excluding the
93590 terminating <newline> matches the given RE. Then, going sequentially from the beginning of
93591 the file to the end of the file, the given *command list* shall be executed for each marked line, with
93592 the current line number set to the address of that line. Any line modified by the *command list*
93593 shall be unmarked. When the **g** command completes, the current line number shall have the
93594 value assigned by the last command in the *command list*. If there were no matching lines, the
93595 current line number shall not be changed. A single command or the first of a list of commands
93596 shall appear on the same line as the global command. All lines of a multi-line list except the last
93597 line shall be ended with a <backslash> preceding the terminating <newline>; the **a**, **i**, and **c**
93598 commands and associated input are permitted. The `'.'` terminating input mode can be omitted
93599 if it would be the last line of the *command list*. An empty *command list* shall be equivalent to the **p**
93600 command. The use of the **g**, **G**, **v**, **V**, and **!** commands in the *command list* produces undefined
93601 results. Any character other than <backslash>, <space>, or <newline> can be used instead of a
93602 <slash> to delimit the RE. Within the RE, in certain circumstances the RE delimiter can be used
93603 as a literal character; see Regular Expressions in ed (on page 2817).

93604 **Interactive Global Command**

93605 *Synopsis*: `(1,$)G/`*RE*`/`

93606 In the **G** command, the first step shall be to mark every line for which the line excluding the
93607 terminating <newline> matches the given RE. Then, for every such line, that line shall be
93608 written, the current line number shall be set to the address of that line, and any one command
93609 (other than one of the **a**, **c**, **i**, **g**, **G**, **v**, and **V** commands) shall be read and executed. A <newline>
93610 shall act as a null command (causing no action to be taken on the current line); an `'&'` shall
93611 cause the re-execution of the most recent non-null command executed within the current
93612 invocation of **G**. Note that the commands input as part of the execution of the **G** command can
93613 address and affect any lines in the buffer. Any line modified by the command shall be
93614 unmarked. The final value of the current line number shall be the value set by the last command
93615 successfully executed. (Note that the last command successfully executed shall be the **G**
93616 command itself if a command fails or the null command is specified.) If there were no matching
93617 lines, the current line number shall not be changed. The **G** command can be terminated by a
93618 SIGINT signal. Any character other than <backslash>, <space>, or <newline> can be used
93619 instead of a <slash> to delimit the RE. Within the RE, in certain circumstances the RE delimiter
93620 can be used as a literal character; see Regular Expressions in ed (on page 2817).

93621 **Help Command**

93622 *Synopsis*: `h`

93623 The **h** command shall write a short message to standard output that explains the reason for the
93624 most recent `'?'` notification. The current line number shall be unchanged.

93625 **Help-Mode Command**

93626 *Synopsis*: `H`

93627 The **H** command shall cause *ed* to enter a mode in which help messages (see the **h** command)
93628 shall be written to standard output for all subsequent `'?'` notifications. The **H** command
93629 alternately shall turn this mode on and off; it is initially off. If the help-mode is being turned on,
93630 the **H** command also explains the previous `'?'` notification, if there was one. The current line
93631 number shall be unchanged.

93632 **Insert Command**

93633 *Synopsis*: `(.)i`
93634 `<`*text*`>`
93635 `.`

93636 The **i** command shall insert the given text before the addressed line; the current line is set to the
93637 last inserted line or, if there was none, to the addressed line. This command differs from the **a**
93638 command only in the placement of the input text. Address 0 shall be valid for this command; it
93639 is unspecified whether it causes the inserted text to be placed at the beginning of the buffer or it
93640 is interpreted as if address 1 were specified. (These two allowed behaviors differ in the case that
93641 the buffer is empty.) If <*text*> is empty (that is, the terminating `'.'` immediately follows the
93642 `'i'`), the buffer change flag shall not be altered.

93643 **Join Command**

93644 *Synopsis*: `(.,.+1)j`

93645 The **j** command shall join contiguous lines by removing the appropriate <newline> characters. If
93646 exactly one address is given, this command shall do nothing. If lines are joined, the current line
93647 number shall be set to the address of the joined line; otherwise, the current line number shall be
93648 unchanged.

93649 **Mark Command**

93650 *Synopsis*: `(.)k`*x*

93651 The **k** command shall mark the addressed line with name *x*, which the application shall ensure
93652 is a lowercase letter from the portable character set. The address `"'x"` shall then refer to this
93653 line; the current line number shall be unchanged.

93654 **List Command**

93655 *Synopsis*: `(.,.)l`

93656 The **l** command shall write to standard output the addressed lines in a visually unambiguous
93657 form. The characters listed in XBD Table 5-1 (on page 113) (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\r'`,
93658 `'\t'`, `'\v'`) shall be written as the corresponding escape sequence; the `'\n'` in that table is not
93659 applicable. Non-printable characters not in the table shall be written as one three-digit octal
93660 number (with a preceding <backslash> character) for each byte in the character (most significant
93661 byte first).

93662 Long lines shall be folded, with the point of folding indicated by <newline> preceded by a
93663 <backslash>; the length at which folding occurs is unspecified, but should be appropriate for the
93664 output device. The end of each line shall be marked with a `'$'`, and `'$'` characters within the
93665 text shall be written with a preceding <backslash>. An **l** command can be appended to any
93666 other command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**. The current line number shall be set to the

93667 address of the last line written.

93668 **Move Command**

93669 *Synopsis*: `(.,.)maddress`

93670 The **m** command shall reposition the addressed lines after the line addressed by *address*.
93671 Address 0 shall be valid for *address* and cause the addressed lines to be moved to the beginning
93672 of the buffer. It shall be an error if address *address* falls within the range of moved lines. The
93673 current line number shall be set to the address of the last line moved.

93674 **Number Command**

93675 *Synopsis*: `(.,.)n`

93676 The **n** command shall write to standard output the addressed lines, preceding each line by its
93677 line number and a <tab>; the current line number shall be set to the address of the last line
93678 written. The **n** command can be appended to any command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

93679 **Print Command**

93680 *Synopsis*: `(.,.)p`

93681 The **p** command shall write to standard output the addressed lines; the current line number shall
93682 be set to the address of the last line written. The **p** command can be appended to any command
93683 other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

93684 **Prompt Command**

93685 *Synopsis*: `P`

93686 The **P** command shall cause *ed* to prompt with an <asterisk> (`'*'`) (or *string*, if −**p** is specified)
93687 for all subsequent commands. The **P** command alternatively shall turn this mode on and off; it
93688 shall be initially on if the −**p** option is specified; otherwise, off. The current line number shall be
93689 unchanged.

93690 **Quit Command**

93691 *Synopsis*: `q`

93692 The **q** command shall cause *ed* to exit. If the buffer change flag is currently set to **changed**, the
93693 user shall be warned, as described previously.

93694 **Quit Without Checking Command**

93695 *Synopsis*: `Q`

93696 The **Q** command shall cause *ed* to exit without checking the current state of the buffer change
93697 flag.

93698 **Read Command**

93699 *Synopsis*: `(`$`)r [`*file*`]`

93700 The **r** command shall read in the file named by the pathname *file* and append it after the
93701 addressed line. If no *file* argument is given, the currently remembered pathname, if any, shall be
93702 used (see the **e** and **f** commands). The currently remembered pathname shall not be changed
93703 unless there is no remembered pathname. Address 0 shall be valid for **r** and shall cause the file
93704 to be read at the beginning of the buffer. If the read is successful, and –**s** was not specified, the
93705 number of bytes read shall be written to standard output in the following format:

93706 `"%d\n",` *<number of bytes read>*

93707 The current line number shall be set to the address of the last line read in. If *file* is replaced by
93708 `'!'`, the rest of the line shall be taken to be a shell command line whose output is to be read.
93709 Such a shell command line shall not be remembered as the current pathname.

93710 If the number of bytes read is 0 it is unspecified whether the buffer change flag is set to **changed**
93711 or left unaltered.

93712 **Substitute Command**

93713 *Synopsis*: `(.,.)s/`*RE*`/`*replacement*`/`*flags*

93714 The **s** command shall search each addressed line for an occurrence of the specified RE and
93715 replace either the first or all (non-overlapped) matched strings with the *replacement*; see the
93716 following description of the **g** suffix. Any character other than <backslash>, <space>, or
93717 <newline> can be used instead of a <slash> to delimit the RE and the replacement. Within the
93718 RE, in certain circumstances the RE delimiter can be used as a literal character; see Regular
93719 Expressions in ed (on page 2817). Within the replacement, the delimiter shall not terminate the
93720 replacement if it is the second character of an escape sequence (see XBD Section 9.1, on page 179)
93721 and the escaped delimiter shall be treated as that literal character in the replacement (losing any
93722 special meaning it would have had if it was not used as the delimiter and was not escaped). It
93723 shall be an error if the substitution fails on every addressed line. The current line shall be set to
93724 the address of the last line on which a substitution occurred.

93725 An unescaped <ampersand> (`'&'`) appearing in the replacement shall be replaced by the string
93726 matching the RE on the current line. As a more general feature, the characters `'\n'`, where the
93727 <backslash> is unescaped and *n* is a digit, shall be replaced by the text matched by the
93728 corresponding back-reference expression. If the corresponding back-reference expression does
93729 not match, then the characters `'\n'` shall be replaced by the empty string. When the character
93730 `'%'` is the only character in *replacement*, the *replacement* used in the most recent substitute
93731 command shall be used as *replacement* in the current substitute command; if there was no
93732 previous substitute command, the use of `'%'` in this manner shall be an error. The `'%'` shall lose
93733 its special meaning when it is in a replacement string of more than one character or is escaped. It
93734 is unspecified what special meaning is given to any character other than <backslash>, `'&'`, `'%'`,
93735 or digits.

93736 A line can be split by substituting a <newline> into it. The application shall ensure it escapes the
93737 <newline> in the *replacement* by preceding it by <backslash>. Such substitution cannot be done
93738 as part of a **g** or **v** *command list*. The current line number shall be set to the address of the last
93739 line on which a substitution is performed. If no substitution is performed, the current line
93740 number shall be unchanged. If a line is split, a substitution shall be considered to have been
93741 performed on each of the new lines for the purpose of determining the new current line number.
93742 A substitution shall be considered to have been performed even if the replacement string is
93743 identical to the string that it replaces.

93744 The application shall ensure that the value of *flags* is zero or more of:

93745 *count* Substitute for the *count*th occurrence only of the RE found on each addressed line.

93746 **g** Globally substitute for all non-overlapping instances of the RE rather than just the first
93747 one. If both **g** and *count* are specified, the results are unspecified.

93748 **l** Write to standard output the final line in which a substitution was made. The line shall
93749 be written in the format specified for the **l** command.

93750 **n** Write to standard output the final line in which a substitution was made. The line shall
93751 be written in the format specified for the **n** command.

93752 **p** Write to standard output the final line in which a substitution was made. The line shall
93753 be written in the format specified for the **p** command.

93754 **Copy Command**

93755 *Synopsis*: `(.,.)t`*`address`*

93756 The **t** command shall be equivalent to the **m** command, except that a copy of the addressed lines
93757 shall be placed after address *address* (which can be 0); the current line number shall be set to the
93758 address of the last line added.

93759 **Undo Command**

93760 *Synopsis*: `u`

93761 The **u** command shall nullify the effect of the most recent command that modified anything in
93762 the buffer, namely the most recent **a**, **c**, **d**, **g**, **i**, **j**, **m**, **r**, **s**, **t**, **u**, **v**, **G**, or **V** command. All changes
93763 made to the buffer by a **g**, **G**, **v**, or **V** global command shall be undone as a single change; if no
93764 changes were made by the global command (such as with **g**/RE/**p**), the **u** command shall have
93765 no effect. The current line number shall be set to the value it had immediately before the
93766 command being undone started.

93767 **Global Non-Matched Command**

93768 *Synopsis*: `(1,$)v/`*`RE`*`/`*`command list`*

93769 This command shall be equivalent to the global command **g** except that the lines that are marked
93770 during the first step shall be those for which the line excluding the terminating <newline> does
93771 not match the RE.

93772 **Interactive Global Not-Matched Command**

93773 *Synopsis*: `(1,$)V/`*`RE`*`/`

93774 This command shall be equivalent to the interactive global command **G** except that the lines that
93775 are marked during the first step shall be those for which the line excluding the terminating
93776 <newline> does not match the RE.

93777 **Write Command**

93778 *Synopsis*: `(1,$)w [`*file*`]`

93779 The **w** command shall write the addressed lines into the file named by the pathname *file*. The
93780 command shall create the file, if it does not exist, or shall replace the contents of the existing file.
93781 The currently remembered pathname shall not be changed unless there is no remembered
93782 pathname. If no pathname is given, the currently remembered pathname, if any, shall be used
93783 (see the **e** and **f** commands); the current line number shall be unchanged. If the command is
93784 successful, the number of bytes written shall be written to standard output, unless the –**s** option
93785 was specified, in the following format:

93786 `"%d\n",` *<number of bytes written>*

93787 If *file* begins with `'!'`, the rest of the line shall be taken to be a shell command line whose
93788 standard input shall be the addressed lines. Such a shell command line shall not be remembered
93789 as the current pathname. This usage of the **w** command with `'!'` shall not alter the buffer
93790 change flag; thus, this alone shall not prevent the warning to the user if an attempt is made to
93791 destroy the editor buffer via the **e** or **q** commands.

93792 **Line Number Command**

93793 *Synopsis*: `($)=`

93794 The line number of the addressed line shall be written to standard output in the following
93795 format:

93796 `"%d\n",` *<line number>*

93797 The current line number shall be unchanged by this command.

93798 **Shell Escape Command**

93799 *Synopsis*: `!`*command*

93800 The remainder of the line after the `'!'` shall be sent to the command interpreter to be
93801 interpreted as a shell command line. Within the text of that shell command line, the unescaped
93802 character `'%'` shall be replaced with the remembered pathname; if a `'!'` appears as the first
93803 character of the command, it shall be replaced with the text of the previous shell command
93804 executed via `'!'`. Thus, `"!!"` shall repeat the previous !*command*. If any replacements of `'%'` or
93805 `'!'` are performed, the modified line shall be written to the standard output before *command* is
93806 executed. The **!** command shall write:

93807 `"!\n"`

93808 to standard output upon completion, unless the –**s** option is specified. The current line number
93809 shall be unchanged.

93810 **Null Command**

93811 *Synopsis*: `(.+1)`

93812 An address alone on a line shall cause the addressed line to be written. A <newline> alone shall
93813 be equivalent to `"+1p"`. The current line number shall be set to the address of the written line.

93814 **EXIT STATUS**

93815 The following exit values shall be returned:

93816 0 Successful completion without any file or command errors.

93817 >0 An error occurred.

93818 **CONSEQUENCES OF ERRORS**

93819 When an error in the input script is encountered, or when an error is detected that is a
93820 consequence of the data (not) present in the file or due to an external condition such as a read or
93821 write error:

93822 • If the standard input is a terminal device file, all input shall be flushed, and a new
93823 command read.

93824 • If the standard input is not a terminal device file, *ed* shall behave as described under
93825 CONSEQUENCES OF ERRORS in Section 1.4 (on page 2462).

93826 **APPLICATION USAGE**

93827 Because of the extremely terse nature of the default error messages, the prudent script writer
93828 begins the *ed* input commands with an **H** command, so that if any errors do occur at least some
93829 clue as to the cause is made available.

93830 In earlier versions of this standard, an obsolescent – option was described. This is no longer
93831 specified. Applications should use the –**s** option. Using – as a *file* operand now produces
93832 unspecified results. This allows implementations to continue to support the former required
93833 behavior.

93834 **EXAMPLES**

93835 None.

93836 **RATIONALE**

93837 The initial description of this utility was adapted from the SVID. It contains some features not
93838 found in Version 7 or BSD-derived systems. Some of the differences between the POSIX and
93839 BSD *ed* utilities include, but need not be limited to:

93840 • The BSD – option does not suppress the `'!'` prompt after a **!** command.

93841 • BSD does not support the special meanings of the `'%'` and `'!'` characters within a **!**
93842 command.

93843 • BSD does not support the *addresses* `';'` and `','`.

93844 • BSD allows the command/suffix pairs **pp**, **ll**, and so on, which are unspecified in this
93845 volume of POSIX.1-2024.

93846 • BSD does not support the `'!'` character part of the **e**, **r**, or **w** commands.

93847 • A failed **g** command in BSD sets the line number to the last line searched if there are no
93848 matches.

93849 • BSD does not default the *command list* to the **p** command.

93850 • BSD does not support the **G**, **h**, **H**, **n**, or **V** commands.

93851 • On BSD, if there is no inserted text, the insert command changes the current line to the
93852 referenced line –1; that is, the line before the specified line.

93853 • On BSD, the **j** command with only a single address changes the current line to that
93854 address.

93855 • BSD does not support the **P** command; moreover, in BSD it is synonymous with the **p**
93856 command.

- BSD does not support the *undo* of the commands **j**, **m**, **r**, **s**, or **t**.
- The Version 7 *ed* command **W**, and the BSD *ed* commands **W**, **wq**, and **z** are not present in this volume of POSIX.1-2024.

The **−s** option was added to allow the functionality of the removed − option in a manner compatible with the Utility Syntax Guidelines.

In early proposals there was a limit, {ED_FILE_MAX}, that described the historical limitations of some *ed* utilities in their handling of large files; some of these have had problems with files larger than 100 000 bytes. It was this limitation that prompted much of the desire to include a *split* command in this volume of POSIX.1-2024. Since this limit was removed, this volume of POSIX.1-2024 requires that implementations document the file size limits imposed by *ed* in the conformance document. The limit {ED_LINE_MAX} was also removed; therefore, the global limit {LINE_MAX} is used for input and output lines.

The manner in which the **l** command writes non-printable characters was changed to avoid the historical backspace-overstrike method. On video display terminals, the overstrike is ambiguous because most terminals simply replace overstruck characters, making the **l** format not useful for its intended purpose of unambiguously understanding the content of the line. The historical <backslash>-escapes were also ambiguous. (The string `"a\0011"` could represent a line containing those six characters or a line containing the three characters `'a'`, a byte with a binary value of 1, and a 1.) In the format required here, a <backslash> appearing in the line is written as `"\\"` so that the output is truly unambiguous. The method of marking the ends of lines was adopted from the *ex* editor and is required for any line ending in <space> characters; the `'$'` is placed on all lines so that a real `'$'` at the end of a line cannot be misinterpreted.

Earlier versions of this standard allowed for implementations with bytes other than eight bits, but this has been modified in this version.

The description of how a NUL is written was removed. The NUL character cannot be in text files, and this volume of POSIX.1-2024 should not dictate behavior in the case of undefined, erroneous input.

Unlike some of the other editing utilities, the filenames accepted by the **E**, **e**, **R**, and **r** commands are not patterns.

Early proposals stated that the **−p** option worked only when standard input was associated with a terminal device. This has been changed to conform to historical implementations, thereby allowing applications to interpose themselves between a user and the *ed* utility.

The form of the substitute command that uses the **n** suffix was limited in some historical documentation (where this was described incorrectly as ``backreferencing''). This limit has been omitted because there is no reason why an editor processing lines of {LINE_MAX} length should have this restriction. The command **s/x/X/2047** should be able to substitute the 2 047th occurrence of `'x'` on a line.

The use of printing commands with printing suffixes (such as **pn**, **lp**, and so on) was made unspecified because BSD-based systems allow this, whereas System V does not.

Some BSD-based systems exit immediately upon receipt of end-of-file if all of the lines in the file have been deleted. Since this volume of POSIX.1-2024 refers to the **q** command in this instance, such behavior is not allowed.

Some historical implementations returned exit status zero even if command errors had occurred; this is not allowed by this volume of POSIX.1-2024.

Some historical implementations contained a bug that allowed a single <period> to be entered in

93902 input mode as <backslash> <period> <newline>. This is not allowed by *ed* because there is no
93903 description of escaping any of the characters in input mode; <backslash> characters are entered
93904 into the buffer exactly as typed. The typical method of entering a single <period> has been to
93905 precede it with another character and then use the substitute command to delete that character.

93906 It is difficult under some modes of some versions of historical operating system terminal drivers
93907 to distinguish between an end-of-file condition and terminal disconnect. POSIX.1-2024 does not
93908 require implementations to distinguish between the two situations, which permits historical
93909 implementations of the *ed* utility on historical platforms to conform. Implementations are
93910 encouraged to distinguish between the two, if possible, and take appropriate action on terminal
93911 disconnect.

93912 Historically, *ed* accepted a zero address for the **a** and **r** commands in order to insert text at the
93913 start of the edit buffer. When the buffer was empty the command **.=** returned zero. POSIX.1-2024
93914 requires conformance to historical practice.

93915 For consistency with the **a** and **r** commands and better user functionality, the **i** command also
93916 accepts an address of 0. However, it is unspecified if `0i` is treated as `1i` (which will fail if the
93917 buffer is empty), or means insert at the beginning of the buffer (which will succeed even if the
93918 buffer is empty). Earlier versions of this standard required address 0 for the **c** command to be
93919 treated as 1 also, but this requirement has been removed, though implementations are permitted
93920 to do this as an extension.

93921 All of the following are valid addresses:

93922 `+++`	Three lines after the current line.
93923 `/`*`pattern`*`/-`	One line before the next occurrence of pattern.
93924 `-2`	Two lines before the current line.
93925 `3 ---- 2`	Line one (note the intermediate negative address).
93926 `1 2 3`	Line six.

93927 More than the maximum number of accepted addresses can be provided to commands taking
93928 addresses; for example, `"1,2,3,4,5p"` prints lines 4 and 5, because two is the maximum
93929 number of addresses accepted by the **print** command. This, in combination with the
93930 <semicolon> delimiter, permits users to create commands based on ordered patterns in the file.
93931 For example, the command `"3;/foo/;+2p"` will display the first line after line 3 that contains
93932 the pattern *foo*, plus the next two lines. Note that the address `"3;"` must still be evaluated before
93933 being discarded, because the search origin for the `"/foo/"` address depends on this.

93934 Historically, *ed* disallowed address chains, as discussed above, consisting solely of <comma> or
93935 <semicolon> separators; for example, `",,,"` or `";;;"` were considered an error. For
93936 consistency of address specification, this restriction is removed. The following table lists some of
93937 the address forms now possible:

Address	Addr1	Addr2	Status	Comment
`7,`	7	7	Historical	
`7,5,`	5	5	Historical	
`7,5,9`	5	9	Historical	
`7,9`	7	9	Historical	
`7,+`	7	`.+`	Historical	
`,`	1	`$`	Historical	
`,7`	1	7	Extension	
`,,`	`$`	`$`	Extension	
`,;`	`$`	`$`	Extension	
`7;`	7	7	Historical	
`7;5;`	5	5	Historical	
`7;5;9`	5	9	Historical	
`7;5,9`	5	9	Historical	
`7;$;4`	`$`	4	Historical	Valid, but erroneous.
`7;9`	7	9	Historical	
`7;+`	7	8	Historical	
`;`	`.`	`$`	Historical	
`;7`	`.`	7	Extension	
`;;`	`$`	`$`	Extension	
`;,`	`$`	`$`	Extension	

Historically, *ed* accepted the `'^'` character as an address, in which case it was identical to the <hyphen-minus> character. POSIX.1-2024 does not require or prohibit this behavior.

Implementations are encouraged to set the buffer change flag to **changed-and-warned** when an **e** or **q** command warns that an attempt was made to destroy the editor buffer. Some existing implementations set it to **unchanged**, but this has the undesirable side-effect that a SIGHUP received after the warning is given does not write the buffer to **ed.hup**.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

A future version of this standard may require that the buffer change flag is set to **changed-and-warned** when an **e** or **q** command warns that an attempt was made to destroy the editor buffer.

SEE ALSO

Section 1.4 (on page 2462), *ex*, *sed*, *sh*, *vi*

XBD Table 5-1 (on page 113), Chapter 8 (on page 167), Section 9.3 (on page 181), Chapter 11 (on page 199), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 5

In the OPTIONS section, the meaning of –**s** and – is clarified.

A second FUTURE DIRECTION is added.

Issue 6

The obsolescent single-minus form is removed.

A second APPLICATION USAGE note is added.

The Open Group Corrigendum U025/2 is applied, correcting the description of the Edit section.

The *ed* utility is updated to align with the IEEE P1003.2b draft standard. This includes addition of the treatment of the SIGQUIT signal, changes to *ed* addressing, and changes to processing when end-of-file is detected and when terminal disconnect is detected.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/22 is applied, adding the text: ``Any line modified by the *command list* shall be unmarked.'' to the **G** command. This change corresponds to a similar change made to the **g** command in the first version of this standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/7 is applied, removing text describing behavior on systems with bytes consisting of more than eight bits.

Issue 7

Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if an operand is `'-'`.

Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.

SD5-XCU-ERN-94 is applied, updating text in the EXTENDED DESCRIPTION where a terminal disconnect is detected (in Commands in *ed*).

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

SD5-XCU-ERN-135 is applied, removing some RATIONALE text that is no longer applicable.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0090 [584], XCU/TC2-2008/0091 [584], and XCU/TC2-2008/0092 [584] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1130 is applied, removing the requirement for the **c** command to accept an address of 0 and updating the information about address 0 in the RATIONALE section.

Austin Group Defect 1131 is applied, changing the address 0 requirements for the **i** command.

Austin Group Defect 1204 is applied, clarifying the behavior when a closing delimiter that would be the last character before a <newline> is omitted.

Austin Group Defect 1281 is applied, moving some text in the description of the **s** command and changing it to use ``shall''.

Austin Group Defect 1298 is applied, changing the CONSEQUENCES OF ERRORS section.

Austin Group Defect 1308 is applied, changing the **Addr2** value for address `7,+` in the table of address forms in the RATIONALE section.

Austin Group Defect 1311 is applied, changing ``*join* command'' to ``**j** command'' in the RATIONALE section.

Austin Group Defect 1582 is applied, clarifying the behavior when an address is omitted between two address separators.

Austin Group Defect 1607 is applied, clarifying the behavior when more than the maximum number of accepted addresses are provided to a command.

Austin Group Defect 1662 is applied, clarifying requirements relating to delimiters in addresses

and in **s** commands.

Austin Group Defect 1786 is applied, clarifying the behavior when an **e** command names a file that does not exist, and clarifying how the *ed* utility keeps track of whether the buffer has been modified.

NAME

env — set the environment for command invocation

SYNOPSIS

```
env [-i] [name=value]... [utility [argument...]]
```

DESCRIPTION

The *env* utility shall obtain the current environment, modify it according to its arguments, then invoke the utility named by the *utility* operand with the modified environment.

Optional arguments shall be passed to *utility*.

If no *utility* operand is specified, the resulting environment shall be written to the standard output, with one *name=value* pair per line.

If the first argument is `'-'`, the results are unspecified.

OPTIONS

The *env* utility shall conform to XBD Section 12.2 (on page 215), except for the unspecified usage of `'-'`.

The following options shall be supported:

-i Invoke *utility* with exactly the environment specified by the arguments; the inherited environment shall be ignored completely.

OPERANDS

The following operands shall be supported:

name=value Arguments of the form *name=value* shall modify the execution environment, and shall be placed into the inherited environment before the *utility* is invoked.

utility The name of the utility to be invoked. If the *utility* operand names any of the special built-in utilities in Section 2.15 (on page 2526), the results are undefined.

argument A string to pass as an argument for the invoked utility.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *env*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

94069 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

94070 *PATH* Determine the location of the *utility*, as described in XBD Chapter 8 (on page 167).
94071 If *PATH* is specified as a *name=value* operand to *env*, the *value* given shall be used in
94072 the search for *utility*.

94073 **ASYNCHRONOUS EVENTS**
94074 Default.

94075 **STDOUT**
94076 If no *utility* operand is specified, each *name=value* pair in the resulting environment shall be
94077 written in the form:

94078 `"%s=%s\n", <name>, <value>`

94079 If the *utility* operand is specified, the *env* utility shall not write to standard output.

94080 **STDERR**
94081 The standard error shall be used only for diagnostic messages.

94082 **OUTPUT FILES**
94083 None.

94084 **EXTENDED DESCRIPTION**
94085 None.

94086 **EXIT STATUS**
94087 If *utility* is invoked, the exit status of *env* shall be the exit status of *utility*; otherwise, the *env*
94088 utility shall exit with one of the following values:

94089 0 The *env* utility completed successfully.

94090 1–125 An error occurred in the *env* utility.

94091 126 The utility specified by *utility* was found but could not be invoked.

94092 127 The utility specified by *utility* could not be found.

94093 **CONSEQUENCES OF ERRORS**
94094 Default.

94095 **APPLICATION USAGE**
94096 The *command*, *env*, *nice*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit
94097 code 127 if a utility to be invoked cannot be found, so that applications can distinguish ``failure
94098 to find a utility'' from ``invoked utility exited with an error indication''. The value 127 was
94099 chosen because it is not commonly used for other meanings; most utilities use small values for
94100 ``normal error conditions'' and the values above 128 can be confused with termination due to
94101 receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could
94102 be found, but not invoked. Some scripts produce meaningful error messages differentiating the
94103 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice
94104 that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any
94105 attempt to *exec* the utility fails for any other reason.

94106 Historical implementations of the *env* utility use the *execvp*( ) or *execlp*( ) functions defined in the
94107 System Interfaces volume of POSIX.1-2024 to invoke the specified utility; this provides better
94108 performance and keeps users from having to escape characters with special meaning to the shell.
94109 Therefore, shell functions, special built-ins, and built-ins that are only provided by the shell are
94110 not found by this type of *env* implementation. However, *env* can be implemented as a shell built-
94111 in, in which case it may be able to execute shell functions and built-ins. An application wishing
94112 to ensure execution of a non-built-in utility can use *exec* in a subshell for this purpose.

EXAMPLES

The following command:

```
env -i PATH=/mybin:"$PATH" $(getconf V7_ENV) mygrep xyz myfile
```

invokes the command *mygrep* with a new *PATH* value as the only entry in its environment other than any variables required by the implementation for conformance. In this case, *PATH* is used to locate *mygrep*, which is expected to reside in **/mybin**.

RATIONALE

As with all other utilities that invoke other utilities, this volume of POSIX.1-2024 only specifies what *env* does with standard input, standard output, standard error, input files, and output files. If a utility is executed, it is not constrained by the specification of input and output by *env*.

The **–i** option was added to allow the functionality of the removed – option in a manner compatible with the Utility Syntax Guidelines. It is possible to create a non-conforming environment using the **–i** option, as it may remove environment variables required by the implementation for conformance. The following will preserve these environment variables as well as preserve the *PATH* for conforming utilities:

```
IFS='
'
# The preceding value should be <space><tab><newline>.
# Set IFS to its default value.

set -f
# disable pathname expansion

\unalias -a
# Unset all possible aliases.
# Note that unalias is escaped to prevent an alias
# being used for unalias.
# This step is not strictly necessary, since aliases are not inherited,
# and the ENV environment variable is only used by interactive shells,
# the only way any aliases can exist in a script is if it defines them
# itself.

unset -f env getconf
# Ensure env and getconf are not user functions.

env -i $(getconf V7_ENV) PATH="$(getconf PATH)" command
```

Some have suggested that *env* is redundant since the same effect is achieved by:

```
name=value ... utility [ argument ... ]
```

The example is equivalent to *env* when an environment variable is being added to the environment of the command, but not when the environment is being set to the given value. The *env* utility also writes out the current environment if invoked without arguments. There is sufficient functionality beyond what the example provides to justify inclusion of *env*.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.15 (on page 2526), Section 2.5 (on page 2478)

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 7

Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first argument is `'−'`.

Austin Group Interpretation 1003.1-2001 #047 is applied, providing RATIONALE on how to use the *env* utility to preserve a conforming environment.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The EXAMPLES section is revised to change the use of *env* **−i**.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1157 is applied, adding a note about shell built-in implementations of *env* to the APPLICATION USAGE section.

Austin Group Defect 1586 is applied, adding the *timeout* utility.

Austin Group Defect 1594 is applied, changing the APPLICATION USAGE section.

NAME

ex — text editor

SYNOPSIS

UP

```
ex [-rR] [-s|-v] [-c command] [-t tagstring] [-w size] [file...]
```

DESCRIPTION

The *ex* utility is a line-oriented text editor. There are two other modes of the editor—open and visual—in which screen-oriented editing is available. This is described more fully by the *ex* **open** and **visual** commands and in *vi*.

If an operand is `'-'`, the results are unspecified.

This section uses the term *edit buffer* to describe the current working text. No specific implementation is implied by this term. All editing changes are performed on the edit buffer, and no changes to it shall affect any file until an editor command writes the file.

Certain terminals do not have all the capabilities necessary to support the complete *ex* definition, such as the full-screen editing commands (*visual mode* or *open mode*). When these commands cannot be supported on such terminals, this condition shall not produce an error message such as ``not an editor command'' or report a syntax error. The implementation may either accept the commands and produce results on the screen that are the result of an unsuccessful attempt to meet the requirements of this volume of POSIX.1-2024 or report an error describing the terminal-related deficiency.

OPTIONS

The *ex* utility shall conform to XBD Section 12.2 (on page 215), except for the unspecified usage of `'-'`, and that `'+'` may be recognized as an option delimiter as well as `'-'`.

The following options shall be supported:

−c *command* Specify an initial command to be executed in the first edit buffer loaded from an existing file (see the EXTENDED DESCRIPTION section). Implementations may support more than a single **−c** option. In such implementations, the specified commands shall be executed in the order specified on the command line.

−r Recover the named files (see the EXTENDED DESCRIPTION section). Recovery information for a file shall be saved during an editor or system crash (for example, when the editor is terminated by a signal which the editor can catch), or after the use of an *ex* **preserve** command.

A *crash* in this context is an unexpected failure of the system or utility that requires restarting the failed system or utility. A system crash implies that any utilities running at the time also crash. In the case of an editor or system crash, the number of changes to the edit buffer (since the most recent **preserve** command) that will be recovered is unspecified.

If no *file* operands are given and the **−t** option is not specified, all other options, the *EXINIT* variable, and any **.exrc** files shall be ignored; a list of all recoverable files available to the invoking user shall be written, and the editor shall exit normally without further action.

−R Set **readonly** edit option.

−s Prepare *ex* for batch use by taking the following actions:

- Suppress writing prompts and informational (but not diagnostic) messages.
- Ignore the value of *TERM* and any implementation default terminal type and assume the terminal is a type incapable of supporting open or visual modes; see the **visual** command and the description of *vi*.
- Suppress the use of the *EXINIT* environment variable and the reading of any **.exrc** file; see the EXTENDED DESCRIPTION section.
- Suppress autoindentation, ignoring the value of the **autoindent** edit option.

−t *tagstring* Edit the file containing the specified *tagstring*; see *ctags*. The tags feature represented by **−t** *tagstring* and the **tag** command is optional. It shall be provided on any system that also provides a conforming implementation of *ctags*; otherwise, the use of **−t** produces undefined results. On any system, it shall be an error to specify more than a single **−t** option.

−v Begin in visual mode (see *vi*).

−w *size* Set the value of the *window* editor option to *size*.

OPERANDS

The following operand shall be supported:

file A pathname of a file to be edited.

STDIN

The standard input consists of a series of commands and input text, as described in the EXTENDED DESCRIPTION section. The implementation may limit each line of standard input to a length of {LINE_MAX}.

If the standard input is not a terminal device, it shall be as if the **−s** option had been specified.

If a read from the standard input returns an error, or if the editor detects an end-of-file condition from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

INPUT FILES

Input files shall be text files or files that would be text files except for an incomplete last line that is not longer than {LINE_MAX}−1 bytes in length and contains no NUL characters. By default, any incomplete last line shall be treated as if it had a trailing <newline>. The editing of other forms of files may optionally be allowed by *ex* implementations.

The **.exrc** files and source files shall be text files consisting of *ex* commands; see the EXTENDED DESCRIPTION section.

By default, the editor shall read lines from the files to be edited without interpreting any of those lines as any form of editor command.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *ex*:

COLUMNS Override the system-selected horizontal screen size. See XBD Chapter 8 (on page 167) for valid values and results when it is unset or null.

EXINIT Determine a list of *ex* commands that are executed on editor start-up. See the EXTENDED DESCRIPTION section for more details of the initialization phase.

HOME Determine a pathname of a directory that shall be searched for an editor start-up file named **.exrc**; see the EXTENDED DESCRIPTION section.

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes within regular expressions, the classification of characters as uppercase or lowercase letters, the case conversion of letters, and the detection of word boundaries.

LC_MESSAGES
Determine the locale used to process affirmative responses, and the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

LINES Override the system-selected vertical screen size, used as the number of lines in a screenful and the vertical screen size in visual mode. See XBD Chapter 8 (on page 167) for valid values and results when it is unset or null.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

PATH Determine the search path for the shell command specified in the *ex* editor commands **!**, **shell**, **read**, and **write**, and the open and visual mode command **!**; see the description of command search and execution in Section 2.9.1.4 (on page 2502).

SHELL Determine the preferred command line interpreter for use as the default value of the **shell** edit option.

TERM Determine the name of the terminal type. If this variable is unset or null, an unspecified default terminal type shall be used.

ASYNCHRONOUS EVENTS

The following term is used in this and following sections to specify command and asynchronous event actions:

complete write
A complete write is a write of the entire contents of the edit buffer to a file of a type other than a terminal device, or the saving of the edit buffer caused by the user executing the *ex* **preserve** command. Writing the contents of the edit buffer to a temporary file that will be removed when the editor exits shall not be considered a complete write.

The following actions shall be taken upon receipt of signals:

SIGINT If the standard input is not a terminal device, *ex* shall not write the file or return to command or text input mode, and shall exit with a non-zero exit status.

Otherwise, if executing an open or visual text input mode command, *ex* in receipt of SIGINT shall behave identically to its receipt of the <ESC> character.

Otherwise:

94297 1. If executing an *ex* text input mode command, all input lines that have been
94298 completely entered shall be resolved into the edit buffer, and any partially
94299 entered line shall be discarded.

94300 2. If there is a currently executing command, it shall be aborted and a message
94301 displayed. Unless otherwise specified by the *ex* or *vi* command descriptions,
94302 it is unspecified whether any lines modified by the executing command
94303 appear modified, or as they were before being modified by the executing
94304 command, in the buffer.

94305 If the currently executing command was a motion command, its associated
94306 command shall be discarded.

94307 3. If in open or visual command mode, the terminal shall be alerted.

94308 4. The editor shall then return to command mode.

94309 SIGCONT If *ex* is in open mode or visual mode, the actions described below for SIGWINCH
94310 shall be taken, except that the screen shall always be refreshed (regardless of
94311 whether the terminal window size changed).

94312 SIGHUP If the edit buffer has been modified since the last complete write, *ex* shall attempt
94313 to save the edit buffer so that it can be recovered later using the **−r** option or the *ex*
94314 **recover** command. The editor shall not write the file or return to command or text
94315 input mode, and shall terminate with a non-zero exit status.

94316 SIGTERM Refer to SIGHUP.

94317 SIGWINCH If *ex* is in open mode or visual mode, the current terminal window size associated
94318 with the terminal on standard output shall be obtained, as if by a call to XSH
94319 *tcgetwinsize*( ). If the terminal window size is successfully obtained, it shall be used
94320 as follows:

94321 • If the *COLUMNS* environment variable is unset or does not contain a
94322 number, the horizontal screen size shall be set to the number of columns in
94323 the obtained terminal window size.

94324 • If *ex* is in visual mode, the **−w** option was not specified and the *LINES*
94325 environment variable is unset or does not contain a number, the vertical
94326 screen size shall be set to the number of rows in the obtained terminal
94327 window size.

94328 If the above resulted in either the vertical screen size or the horizontal screen size
94329 (or both) changing to a different value, *ex* shall update the values it has for the
94330 number of lines and columns in the display and shall adjust the **window** edit
94331 option and the column number at which the **wrapmargin** edit option takes effect
94332 (if non-zero) accordingly (see Edit Options in ex, on page 2877) and refresh the
94333 screen; otherwise, *ex* may refresh the screen.

94334 The action taken for all other signals is unspecified.

94335 **STDOUT**

94336 The standard output shall be used only for writing prompts to the user, for informational
94337 messages, and for writing lines from the file.

94338 **STDERR**

94339 The standard error shall be used only for diagnostic messages.

94340 **OUTPUT FILES**

94341 The output from *ex* shall be text files.

94342 **EXTENDED DESCRIPTION**

94343 Only the *ex* mode of the editor is described in this section. See *vi* for additional editing
94344 capabilities available in *ex*.

94345 When an error occurs, *ex* shall write a message. If the terminal supports a standout mode (such
94346 as inverse video), the message shall be written in standout mode. If the terminal does not
94347 support a standout mode, and the edit option **errorbells** is set, an alert action shall precede the
94348 error message.

94349 By default, *ex* shall start in command mode, which shall be indicated by a **:** prompt; see the
94350 **prompt** command. Text input mode can be entered by the **append**, **insert**, or **change** commands;
94351 it can be exited (and command mode re-entered) by typing a <period> (`'.'`) alone at the
94352 beginning of a line.

94353 **Initialization in ex and vi**

94354 The following symbols are used in this and following sections to specify locations in the edit
94355 buffer:

94356 *alternate and current pathnames*

94357 Two pathnames, named *current* and *alternate*, are maintained by the editor. Any *ex*
94358 commands that take filenames as arguments shall set them as follows:

94359 1. If a *file* argument is specified to the *ex* **edit**, **ex**, or **recover** commands, or if an *ex* **tag**
94360 command replaces the contents of the edit buffer.

94361 a. If the command replaces the contents of the edit buffer, the current pathname
94362 shall be set to the *file* argument or the file indicated by the tag, and the
94363 alternate pathname shall be set to the previous value of the current pathname.

94364 b. Otherwise, the alternate pathname shall be set to the *file* argument.

94365 2. If a *file* argument is specified to the *ex* **next** command:

94366 a. If the command replaces the contents of the edit buffer, the current pathname
94367 shall be set to the first *file* argument, and the alternate pathname shall be set to
94368 the previous value of the current pathname.

94369 3. If a *file* argument is specified to the *ex* **file** command, the current pathname shall be
94370 set to the *file* argument, and the alternate pathname shall be set to the previous value
94371 of the current pathname.

94372 4. If a *file* argument is specified to the *ex* **read** and **write** commands (that is, when
94373 reading or writing a file, and not to the program named by the **shell** edit option), or a
94374 *file* argument is specified to the *ex* **xit** command:

94375 a. If the current pathname has no value, the current pathname shall be set to the
94376 *file* argument.

94377 b. Otherwise, the alternate pathname shall be set to the *file* argument.

94378 If the alternate pathname is set to the previous value of the current pathname when the
94379 current pathname had no previous value, then the alternate pathname shall have no value
94380 as a result.

current line
: The line of the edit buffer referenced by the cursor. Each command description specifies the current line after the command has been executed, as the *current line value*. When the edit buffer contains no lines, the current line shall be zero; see Addressing in ex (on page 2845).

current column
: The current display line column occupied by the cursor. (The columns shall be numbered beginning at 1.) Each command description specifies the current column after the command has been executed, as the *current column* value. This column is an *ideal* column that is remembered over the lifetime of the editor. The actual display line column upon which the cursor rests may be different from the current column; see the cursor positioning discussion in Command Descriptions in vi (on page 3527).

set to non-<blank>
: A description for a current column value, meaning that the current column shall be set to the last display line column on which is displayed any part of the first non-<blank> of the line. If the line has no non-<blank> non-<newline> characters, the current column shall be set to the last display line column on which is displayed any part of the last non-<newline> character in the line. If the line is empty, the current column shall be set to column position 1.

The length of lines in the edit buffer may be limited to {LINE_MAX} bytes. In open and visual mode, the length of lines in the edit buffer may be limited to the number of characters that will fit in the display. If either limit is exceeded during editing, an error message shall be written. If either limit is exceeded by a line read in from a file, an error message shall be written and the edit session may be terminated.

If the editor stops running due to any reason other than a user command, and the edit buffer has been modified since the last complete write, it shall be equivalent to a SIGHUP asynchronous event. If the system crashes, it shall be equivalent to a SIGHUP asynchronous event.

During initialization (before the first file is copied into the edit buffer or any user commands from the terminal are processed) the following shall occur:

1. If the environment variable *EXINIT* is set, the editor shall execute the *ex* commands contained in that variable.
2. If the *EXINIT* variable is not set, and all of the following are true:
    a. The *HOME* environment variable is not null and not empty.
    b. The file **.exrc** in the directory referred to by the *HOME* environment variable:
        i. Exists
        ii. Is owned by the same user ID as the real user ID of the process or the process has appropriate privileges
        iii. Is not writable by anyone other than the owner

    the editor shall execute the *ex* commands contained in that file.
3. If and only if all of the following are true:
    a. The current directory is not referred to by the *HOME* environment variable.
    b. A command in the *EXINIT* environment variable or a command in the **.exrc** file in the directory referred to by the *HOME* environment variable sets the editor option **exrc**.

94424 c. The **.exrc** file in the current directory:

94425 i. Exists

94426 ii. Is owned by the same user ID as the real user ID of the process, or by one of
94427 a set of implementation-defined user IDs

94428 iii. Is not writable by anyone other than the owner

94429 the editor shall attempt to execute the *ex* commands contained in that file.

94430 Lines in any **.exrc** file that are blank lines shall be ignored. If any **.exrc** file exists, but is not read
94431 for ownership or permission reasons, it shall be an error.

94432 After the *EXINIT* variable and any **.exrc** files are processed, the first file specified by the user
94433 shall be edited, as follows:

94434 1. If the user specified the −**t** option, the effect shall be as if the *ex* **tag** command was entered
94435 with the specified argument, with the exception that if tag processing does not result in a
94436 file to edit, the effect shall be as described in step 3. below.

94437 2. Otherwise, if the user specified any command line *file* arguments, the effect shall be as if
94438 the *ex* **edit** command was entered with the first of those arguments as its *file* argument.

94439 3. Otherwise, the effect shall be as if the *ex* **edit** command was entered with a nonexistent
94440 filename as its *file* argument. It is unspecified whether this action shall set the current
94441 pathname. In an implementation where this action does not set the current pathname, any
94442 editor command using the current pathname shall fail until an editor command sets the
94443 current pathname.

94444 If the −**r** option was specified, the first time a file in the initial argument list or a file specified by
94445 the −**t** option is edited, if recovery information has previously been saved about it, that
94446 information shall be recovered and the editor shall behave as if the contents of the edit buffer
94447 have already been modified. If there are multiple instances of the file to be recovered, the one
94448 most recently saved shall be recovered, and an informational message that there are previous
94449 versions of the file that can be recovered shall be written. If no recovery information about a file
94450 is available, an informational message to this effect shall be written, and the edit shall proceed as
94451 usual.

94452 If the −**c** option was specified, the first time a file that already exists (including a file that might
94453 not exist but for which recovery information is available, when the −**r** option is specified)
94454 replaces or initializes the contents of the edit buffer, the current line shall be set to the last line of
94455 the edit buffer, the current column shall be set to non-<blank>, and the *ex* commands specified
94456 with the −**c** option shall be executed. In this case, the current line and current column shall not
94457 be set as described for the command associated with the replacement or initialization of the edit
94458 buffer contents. However, if the −**t** option or a **tag** command is associated with this action, the −**c**
94459 option commands shall be executed and then the movement to the tag shall be performed.

94460 The current argument list shall initially be set to the filenames specified by the user on the
94461 command line. If no filenames are specified by the user, the current argument list shall be empty.
94462 If the −**t** option was specified, it is unspecified whether any filename resulting from tag
94463 processing shall be prepended to the current argument list. In the case where the filename is
94464 added as a prefix to the current argument list, the current argument list reference shall be set to
94465 that filename. In the case where the filename is not added as a prefix to the current argument
94466 list, the current argument list reference shall logically be located before the first of the filenames
94467 specified on the command line (for example, a subsequent *ex* **next** command shall edit the first
94468 filename from the command line). If the −**t** option was not specified, the current argument list
94469 reference shall be to the first of the filenames on the command line.

94470 **Addressing in ex**

94471 Addressing in *ex* relates to the current line and the current column; the address of a line is its
94472 1-based line number, the address of a column is its 1-based count from the beginning of the line.
94473 Generally, the current line is the last line affected by a command. The current line number is the
94474 address of the current line. In each command description, the effect of the command on the
94475 current line number and the current column is described.

94476 Addresses are constructed as follows:

94477 1. The character `'.'` (period) shall address the current line.

94478 2. The character `'$'` shall address the last line of the edit buffer.

94479 3. The positive decimal number *n* shall address the *n*th line of the edit buffer.

94480 4. The address `"'x"` refers to the line marked with the mark name character `'x'`, which
94481 shall be a lowercase letter from the portable character set, the backquote character, or the
94482 single-quote character. It shall be an error if the line that was marked is not currently
94483 present in the edit buffer or the mark has not been set. Lines can be marked with the *ex*
94484 **mark** or **k** commands, or the *vi* **m** command.

94485 5. A regular expression enclosed by <slash> characters (`'/'`) shall address the first line
94486 found by searching forwards from the line following the current line toward the end of
94487 the edit buffer and stopping at the first line for which the line excluding the terminating
94488 <newline> matches the regular expression. As stated in Regular Expressions in ex (on
94489 page 2875), an address consisting of a null regular expression delimited by <slash>
94490 characters (`"//"`) shall address the next line for which the line excluding the terminating
94491 <newline> matches the last regular expression encountered. In addition, the second
94492 <slash> can be omitted at the end of a command line. If the **wrapscan** edit option is set,
94493 the search shall wrap around to the beginning of the edit buffer and continue up to and
94494 including the current line, so that the entire edit buffer is searched. Within the regular
94495 expression, the sequence `"\/"` shall represent a literal <slash> instead of the regular
94496 expression delimiter.

94497 6. A regular expression enclosed in <question-mark> characters (`'?'`) shall address the first
94498 line found by searching backwards from the line preceding the current line toward the
94499 beginning of the edit buffer and stopping at the first line for which the line excluding the
94500 terminating <newline> matches the regular expression. An address consisting of a null
94501 regular expression delimited by <question-mark> characters (`"??"`) shall address the
94502 previous line for which the line excluding the terminating <newline> matches the last
94503 regular expression encountered. In addition, the second <question-mark> can be omitted
94504 at the end of a command line. If the **wrapscan** edit option is set, the search shall wrap
94505 around from the beginning of the edit buffer to the end of the edit buffer and continue up
94506 to and including the current line, so that the entire edit buffer is searched. Within the
94507 regular expression, the sequence `"\?"` shall represent a literal <question-mark> instead
94508 of the RE delimiter.

94509 7. A <plus-sign> (`'+'`) or a <hyphen-minus> (`'−'`) followed by a decimal number shall
94510 address the current line plus or minus the number. A `'+'` or `'−'` not followed by a
94511 decimal number shall address the current line plus or minus 1.

94512 Addresses can be followed by zero or more address offsets, optionally <blank>-separated.
94513 Address offsets are constructed as follows:

94514 1. A '+' or '−' immediately followed by a decimal number shall add (subtract) the
94515 indicated number of lines to (from) the address. A '+' or '−' not followed by a decimal
94516 number shall add (subtract) 1 to (from) the address.

94517 2. A decimal number shall add the indicated number of lines to the address.

94518 It shall not be an error for an intermediate address value to be less than zero or greater than the
94519 last line in the edit buffer. It shall be an error for the final address value to be less than zero or
94520 greater than the last line in the edit buffer.

94521 Commands take zero, one, or two addresses; see the descriptions of *1addr* and *2addr* in
94522 Command Descriptions in ex (on page 2852). If more than the required number of addresses are
94523 provided to a command that requires zero addresses, it shall be an error. Otherwise, if more than
94524 the required number of addresses are provided to a command, the addresses specified first shall
94525 be evaluated and then discarded until the maximum number of valid addresses remain.

94526 Addresses shall be separated from each other by a <comma> (',') or a <semicolon> (';'). If
94527 no address is specified before or after a <comma> or <semicolon> separator, it shall be as if the
94528 address of the current line was specified before or after the separator. In the case of a
94529 <semicolon> separator, the current line ('.') shall be set to the first address, and only then shall
94530 the next address be calculated. This feature can be used to determine the starting line for
94531 forwards and backwards searches (see rules 5. and 6.).

94532 A <percent-sign> ('%') shall be equivalent to entering the two addresses "1,$".

94533 Any delimiting <blank> characters between addresses, address separators, or address offsets
94534 shall be discarded.

94535 **Command Line Parsing in ex**

94536 The following symbol is used in this and following sections to describe parsing behavior:

94537 *escape* If a character is referred to as ``<backslash>-escaped'' or ``<control>-V-escaped'', it
94538 shall mean that the character acquired or lost a special meaning by virtue of being
94539 preceded, respectively, by a <backslash> or <control>-V character. Unless
94540 otherwise specified, the escaping character shall be discarded at that time and shall
94541 not be further considered for any purpose.

94542 Command-line parsing shall be done in the following steps. For each step, characters already
94543 evaluated shall be ignored; that is, the phrase ``leading character'' refers to the next character
94544 that has not yet been evaluated.

94545 1. Leading <colon> characters shall be skipped.

94546 2. Leading <blank> characters shall be skipped.

94547 3. If the leading character is a double-quote character, the characters up to and including the
94548 next non-<backslash>-escaped <newline> shall be discarded, and any subsequent
94549 characters shall be parsed as a separate command.

94550 4. Leading characters that can be interpreted as addresses shall be evaluated; see
94551 Addressing in ex (on page 2845).

94552 5. Leading <blank> characters shall be skipped.

94553 6. If the next character is a <vertical-line> character or a <newline>:

94554 a. If the next character is a <newline>:

94555 94556 i. If *ex* is in open or visual mode, the current line shall be set to the last address specified, if any.

94557 94558 94559 ii. Otherwise, if the last command was terminated by a <vertical-line> character, no action shall be taken; for example, the command "`||<newline>`" shall execute two implied commands, not three.

94560 iii. Otherwise, step 6.b. shall apply.

94561 94562 94563 94564 94565 b. Otherwise, the implied command shall be the **print** command. The last **#**, **p**, and **l** flags specified to any *ex* command shall be remembered and shall apply to this implied command. Executing the *ex* **number**, **print**, or **list** command shall set the remembered flags to **#**, nothing, and **l**, respectively, plus any other flags specified for that execution of the **number**, **print**, or **list** command.

94566 94567 94568 94569 If *ex* is not currently performing a **global** or **v** command, and no address or count is specified, the current line shall be incremented by 1 before the command is executed. If incrementing the current line would result in an address past the last line in the edit buffer, the command shall fail, and the increment shall not happen.

94570 94571 c. The <newline> or <vertical-line> character shall be discarded and any subsequent characters shall be parsed as a separate command.

94572 94573 94574 7. The command name shall be comprised of the next character (if the character is not alphabetic), or the next character and any subsequent alphabetic characters (if the character is alphabetic), with the following exceptions:

94575 94576 94577 94578 94579 94580 a. Commands that consist of any prefix of the characters in the command name **delete**, followed immediately by any of the characters '`l`', '`p`', '`+`', '`-`', or '`#`' shall be interpreted as a **delete** command, followed by a <blank>, followed by the characters that were not part of the prefix of the **delete** command. The maximum number of characters shall be matched to the command name **delete**; for example, "`del`" shall not be treated as "`de`" followed by the flag **l**.

94581 94582 94583 b. Commands that consist of the character '`k`', followed by a character that can be used as the name of a mark, shall be equivalent to the mark command followed by a <blank>, followed by the character that followed the '`k`'.

94584 94585 94586 94587 c. Commands that consist of the character '`s`', followed by characters that could be interpreted as valid options to the **s** command, shall be the equivalent of the **s** command, without any pattern or replacement values, followed by a <blank>, followed by the characters after the '`s`'.

94588 94589 94590 94591 8. The command name shall be matched against the possible command names, and a command name that contains a prefix matching the characters specified by the user shall be the executed command. In the case of commands where the characters specified by the user could be ambiguous, the executed command shall be as follows:

94592 94593 94594 94595 94596 94597

a	**append**	**n**	**next**	**t**	**t**
c	**change**	**p**	**print**	**u**	**undo**
ch	**change**	**pr**	**print**	**un**	**undo**
e	**edit**	**r**	**read**	**v**	**v**
m	**move**	**re**	**read**	**w**	**write**
ma	**mark**	**s**	**s**		

94598 94599 Implementation extensions with names causing similar ambiguities shall not be checked for a match until all possible matches for commands specified by POSIX.1-2024 have been

94600 checked.

94601 9. If the command is a **!** command, or if the command is a **read** command followed by zero
94602 or more <blank> characters and a **!**, or if the command is a **write** command followed by
94603 one or more <blank> characters and a **!**, the rest of the command shall include all
94604 characters up to a non-<backslash>-escaped <newline>. The <newline> shall be
94605 discarded and any subsequent characters shall be parsed as a separate *ex* command.

94606 10. Otherwise, if the command is an **edit**, **ex**, or **next** command, or a **visual** command while
94607 in open or visual mode, the next part of the command shall be parsed as follows:

94608 a. Any `'!'` character immediately following the command shall be skipped and be
94609 part of the command.

94610 b. Any leading <blank> characters shall be skipped and be part of the command.

94611 c. If the next character is a `'+'`, characters up to the first non-<backslash>-escaped
94612 <newline> or non-<backslash>-escaped <blank> shall be skipped and be part of
94613 the command.

94614 d. The rest of the command shall be determined by the steps specified in paragraph
94615 12.

94616 11. Otherwise, if the command is a **global**, **open**, **s**, or **v** command, the next part of the
94617 command shall be parsed as follows:

94618 a. Any leading <blank> characters shall be skipped and be part of the command.

94619 b. If the next character is not an alphanumeric, double-quote, <newline>,
94620 <backslash>, or <vertical-line> character:

94621 i. The next character shall be used as a command delimiter.

94622 ii. If the command is a **global**, **open**, or **v** command, characters up to the first
94623 non-<backslash>-escaped <newline>, or first non-<backslash>-escaped
94624 delimiter character, shall be skipped and be part of the command.

94625 iii. If the command is an **s** command, characters up to the first
94626 non-<backslash>-escaped <newline>, or second non-<backslash>-escaped
94627 delimiter character, shall be skipped and be part of the command.

94628 c. If the command is a **global** or **v** command, characters up to the first
94629 non-<backslash>-escaped <newline> shall be skipped and be part of the
94630 command.

94631 d. Otherwise, the rest of the command shall be determined by the steps specified in
94632 paragraph 12.

94633 12. Otherwise:

94634 a. If the command was a **map**, **unmap**, **abbreviate**, or **unabbreviate** command,
94635 characters up to the first non-<control>-V-escaped <newline>, <vertical-line>, or
94636 double-quote character shall be skipped and be part of the command.

94637 b. Otherwise, characters up to the first non-<backslash>-escaped <newline>,
94638 <vertical-line>, or double-quote character shall be skipped and be part of the
94639 command.

94640 c. If the command was an **append**, **change**, or **insert** command, and the step 12.b.
94641 ended at a <vertical-line> character, any subsequent characters, up to the next
94642 non-<backslash>-escaped <newline> shall be used as input text to the command.

94643 d. If the command was ended by a double-quote character, all subsequent characters,
94644 up to the next non-<backslash>-escaped <newline>, shall be discarded.

94645 e. The terminating <newline> or <vertical-line> character shall be discarded and any
94646 subsequent characters shall be parsed as a separate *ex* command.

94647 Command arguments shall be parsed as described by the Synopsis and Description of each
94648 individual *ex* command. This parsing shall not be <blank>-sensitive, except for the **!** argument,
94649 which has to follow the command name without intervening <blank> characters, and where it
94650 would otherwise be ambiguous. For example, *count* and *flag* arguments need not be
94651 <blank>-separated because `"d22p"` is not ambiguous, but *file* arguments to the *ex* **next**
94652 command need to be separated by one or more <blank> characters. Any <blank> in command
94653 arguments for the **abbreviate**, **unabbreviate**, **map**, and **unmap** commands can be
94654 <control>-V-escaped, in which case the <blank> shall not be used as an argument delimiter. Any
94655 <blank> in the command argument for any other command can be <backslash>-escaped, in
94656 which case that <blank> shall not be used as an argument delimiter.

94657 Within command arguments for the **abbreviate**, **unabbreviate**, **map**, and **unmap** commands,
94658 any character can be <control>-V-escaped. All such escaped characters shall be treated literally
94659 and shall have no special meaning. Within command arguments for all other *ex* commands that
94660 are not regular expressions or replacement strings, any character that would otherwise have a
94661 special meaning can be <backslash>-escaped. Escaped characters shall be treated literally,
94662 without special meaning as shell expansion characters or `'!'`, `'%'`, and `'#'` expansion
94663 characters. See Regular Expressions in ex (on page 2875) and Replacement Strings in ex (on page
94664 2876) for descriptions of command arguments that are regular expressions or replacement
94665 strings.

94666 Non-<backslash>-escaped `'%'` characters appearing in *file* arguments to any *ex* command shall
94667 be replaced by the current pathname; unescaped `'#'` characters shall be replaced by the
94668 alternate pathname. It shall be an error if `'%'` or `'#'` characters appear unescaped in an
94669 argument and their corresponding values are not set.

94670 Non-<backslash>-escaped `'!'` characters in the arguments to either the *ex* **!** command or the
94671 open and visual mode **!** command, or in the arguments to the *ex* **read** command, where the first
94672 non-<blank> after the command name is a `'!'` character, or in the arguments to the *ex* **write**
94673 command where the command name is followed by one or more <blank> characters and the
94674 first non-<blank> after the command name is a `'!'` character, shall be replaced with the
94675 arguments to the last of those three commands as they appeared after all unescaped `'%'`, `'#'`,
94676 and `'!'` characters were replaced. It shall be an error if `'!'` characters appear unescaped in one
94677 of these commands and there has been no previous execution of one of these commands.

94678 If an error occurs during the parsing or execution of an *ex* command:

94679 • An informational message to this effect shall be written. Execution of the *ex* command shall
94680 stop, and the cursor (for example, the current line and column) shall not be further
94681 modified.

94682 • If the *ex* command resulted from a map expansion, all characters from that map expansion
94683 shall be discarded, except as otherwise specified by the **map** command.

94684 • Otherwise, if the *ex* command resulted from the processing of an *EXINIT* environment
94685 variable, a **.exrc** file, a **:source** command, a **–c** option, or a **+***command* specified to an *ex* **edit**,
94686 **ex**, **next**, or **visual** command, no further commands from the source of the commands shall
94687 be executed.

94688 • Otherwise, if the *ex* command resulted from the execution of a buffer or a **global** or **v**
94689 command, no further commands caused by the execution of the buffer or the **global** or **v**
94690 command shall be executed.

94691 • Otherwise, if the *ex* command was not terminated by a <newline>, all characters up to and
94692 including the next non-<backslash>-escaped <newline> shall be discarded.

94693 **Input Editing in ex**

94694 The following symbol is used in this and the following sections to specify command actions:

94695 *word* In the POSIX locale, a word consists of a maximal sequence of letters, digits, and
94696 underscores, delimited at both ends by characters other than letters, digits, or
94697 underscores, or by the beginning or end of a line or the edit buffer.

94698 When accepting input characters from the user, in either *ex* command mode or *ex* text input
94699 mode, *ex* shall enable canonical mode input processing, as defined in the System Interfaces
94700 volume of POSIX.1-2024.

94701 If in *ex* text input mode:

94702 1. If the **number** edit option is set, *ex* shall prompt for input using the line number that
94703 would be assigned to the line if it is entered, in the format specified for the *ex* **number**
94704 command.

94705 2. If the **autoindent** edit option is set, *ex* shall prompt for input using **autoindent** characters,
94706 as described by the **autoindent** edit option. **autoindent** characters shall follow the line
94707 number, if any.

94708 If in *ex* command mode:

94709 1. If the **prompt** edit option is set, input shall be prompted for using a single `':'` character;
94710 otherwise, there shall be no prompt.

94711 The input characters in the following sections shall have the following effects on the input line.

94712 **Scroll**

94713 *Synopsis*: `eof`

94714 See the description of the *stty eof* character in *stty*.

94715 If in *ex* command mode:

94716 If the *eof* character is the first character entered on the line, the line shall be evaluated as if
94717 it contained two characters: a <control>-D and a <newline>.

94718 Otherwise, the *eof* character shall have no special meaning.

94719 If in *ex* text input mode:

94720 If the cursor follows an **autoindent** character, the **autoindent** characters in the line shall be
94721 modified so that a part of the next text input character is displayed on the first column in
94722 the line after the previous **shiftwidth** edit option column boundary, and the user shall be
94723 prompted again for input for the same line.

94724 Otherwise, if the cursor follows a `'0'`, which follows an **autoindent** character, and the `'0'`
94725 was the previous text input character, the `'0'` and all **autoindent** characters in the line
94726 shall be discarded, and the user shall be prompted again for input for the same line.

94727 Otherwise, if the cursor follows a `'^'`, which follows an **autoindent** character, and the `'^'`
94728 was the previous text input character, the `'^'` and all **autoindent** characters in the line
94729 shall be discarded, and the user shall be prompted again for input for the same line. In
94730 addition, the **autoindent** level for the next input line shall be derived from the same line
94731 from which the **autoindent** level for the current input line was derived.

94732 Otherwise, if there are no **autoindent** or text input characters in the line, the *eof* character
94733 shall be discarded.

94734 Otherwise, the *eof* character shall have no special meaning.

94735 **<newline>**

94736 *Synopsis*: `<newline>`
94737 `<control>-J`

94738 If in *ex* command mode:

94739 Cause the command line to be parsed; <control>-J shall be mapped to the <newline> for
94740 this purpose.

94741 If in *ex* text input mode:

94742 Terminate the current line. If there are no characters other than **autoindent** characters on
94743 the line, all characters on the line shall be discarded.

94744 Prompt for text input on a new line after the current line. If the **autoindent** edit option is
94745 set, an appropriate number of **autoindent** characters shall be added as a prefix to the line
94746 as described by the *ex* **autoindent** edit option.

94747 **<backslash>**

94748 *Synopsis*: `<backslash>`

94749 Allow the entry of a subsequent <newline> or <control>-J as a literal character, removing any
94750 special meaning that it may have to the editor during text input mode. The <backslash>
94751 character shall be retained and evaluated when the command line is parsed, or retained and
94752 included when the input text becomes part of the edit buffer.

94753 **<control>-V**

94754 *Synopsis*: `<control>-V`

94755 Allow the entry of any subsequent character as a literal character, removing any special meaning
94756 that it may have to the editor during text input mode. The <control>-V character shall be
94757 discarded before the command line is parsed or the input text becomes part of the edit buffer.

94758 If the ``literal next'' functionality is performed by the underlying system, it is implementation-
94759 defined whether a character other than <control>-V performs this function.

94760 **<control>-W**

94761 *Synopsis*: `<control>-W`

94762 Discard the <control>-W, and the word previous to it in the input line, including any <blank>
94763 characters following the word and preceding the <control>-W. If the ``word erase'' functionality
94764 is performed by the underlying system, it is implementation-defined whether a character other
94765 than <control>-W performs this function.

94766 **Command Descriptions in ex**

94767 The following symbols are used in this section to represent command modifiers. Some of these
94768 modifiers can be omitted, in which case the specified defaults shall be used.

94769 *1addr* A single line address, given in any of the forms described in Addressing in ex (on
94770 page 2845); the default shall be the current line (`'.'`), unless otherwise specified.

94771 If the line address is zero, it shall be an error, unless otherwise specified in the
94772 following command descriptions.

94773 If the edit buffer is empty, and the address is specified with a command other than
94774 **=**, **append**, **insert**, **open**, **put**, **read**, or **visual**, or the address is not zero, it shall be
94775 an error.

94776 *2addr* Two addresses specifying an inclusive range of lines. If no addresses are specified,
94777 the default for *2addr* shall be the current line only (`"."`), unless otherwise
94778 specified in the following command descriptions. If one address is specified, *2addr*
94779 shall specify that line only, unless otherwise specified in the following command
94780 descriptions.

94781 It shall be an error if the first address is greater than the second address.

94782 If the edit buffer is empty, and the two addresses are specified with a command
94783 other than the **!**, **write**, **wq**, or **xit** commands, or either address is not zero, it shall
94784 be an error.

94785 *count* A positive decimal number. If *count* is specified, it shall be equivalent to specifying
94786 an additional address to the command, unless otherwise specified by the following
94787 command descriptions. The additional address shall be equal to the last address
94788 specified to the command (either explicitly or by default) plus *count*–1.

94789 If this would result in an address greater than the last line of the edit buffer, it shall
94790 be corrected to equal the last line of the edit buffer.

94791 *flags* One or more of the characters `'+'`, `'-'`, `'#'`, `'p'`, or `'l'` (ell). The flag characters
94792 can be <blank>-separated, and in any order or combination. The characters `'#'`,
94793 `'p'`, and `'l'` shall cause lines to be written in the format specified by the **print**
94794 command with the specified *flags*.

94795 The lines to be written are as follows:

94796 1. All edit buffer lines written during the execution of the *ex* **&**, **~**, **list**, **number**,
94797 **open**, **print**, **s**, **visual**, and **z** commands shall be written as specified by *flags*.

94798 2. After the completion of an *ex* command with a flag as an argument, the
94799 current line shall be written as specified by *flags*, unless the current line was
94800 the last line written by the command.

94801 The characters `'+'` and `'−'` cause the value of the current line after the execution
94802 of the *ex* command to be adjusted by the offset address as described in Addressing
94803 in ex (on page 2845). This adjustment shall occur before the current line is written
94804 as described in 2. above.

94805 The default for *flags* shall be none.

94806 *buffer* One of a number of named areas for holding text. The named buffers are specified
94807 by the alphanumeric characters of the POSIX locale. There shall also be one
94808 ``unnamed'' buffer. When no buffer is specified for editor commands that use a
94809 buffer, the unnamed buffer shall be used. Commands that store text into buffers
94810 shall store the text as it was before the command took effect, and shall store text
94811 occurring earlier in the file before text occurring later in the file, regardless of how
94812 the text region was specified. Commands that store text into buffers shall store the
94813 text into the unnamed buffer as well as any specified buffer.

94814 In *ex* commands, buffer names are specified as the name by itself. In open or visual
94815 mode commands the name is preceded by a double-quote (`'"'`) character.

94816 If the specified buffer name is an uppercase character, and the buffer contents are
94817 to be modified, the buffer shall be appended to rather than being overwritten. If
94818 the buffer is not being modified, specifying the buffer name in lowercase and
94819 uppercase shall have identical results.

94820 There shall also be buffers named by the numbers 1 through 9. In open and visual
94821 mode, if a region of text including characters from more than a single line is being
94822 modified by the *vi* **c** or **d** commands, the motion character associated with the **c** or
94823 **d** commands specifies that the buffer text shall be in line mode, or the commands
94824 **%**, **`**, **/**, **?**, **(**, **)**, **N**, **n**, **{**, or **}** are used to define a region of text for the **c** or **d** commands,
94825 the contents of buffers 1 through 8 shall be moved into the buffer named by the
94826 next numerically greater value, the contents of buffer 9 shall be discarded, and the
94827 region of text shall be copied into buffer 1. This shall be in addition to copying the
94828 text into a user-specified buffer or unnamed buffer, or both. Numeric buffers can
94829 be specified as a source buffer for open and visual mode commands; however,
94830 specifying a numeric buffer as the write target of an open or visual mode
94831 command shall have unspecified results.

94832 The text of each buffer shall have the characteristic of being in either line or
94833 character mode. Appending text to a non-empty buffer shall set the mode to match
94834 the characteristic of the text being appended. Appending text to a buffer shall
94835 cause the creation of at least one additional line in the buffer. All text stored into
94836 buffers by *ex* commands shall be in line mode. The *ex* commands that use buffers
94837 as the source of text specify individually how buffers of different modes are
94838 handled. Each open or visual mode command that uses buffers for any purpose
94839 specifies individually the mode of the text stored into the buffer and how buffers
94840 of different modes are handled.

94841 *file* Command text used to derive a pathname. The default shall be the current
94842 pathname, as defined previously, in which case, if no current pathname has yet
94843 been established it shall be an error, except where specifically noted in the
94844 individual command descriptions that follow. If the command text contains any of
94845 the characters `'~'`, `'{'`, `'['`, `'*'`, `'?'`, `'$'`, `'"'`, backquote, single-quote, and
94846 <backslash>, it shall be subjected to the process of ``shell expansions'', as described
94847 below; if more than a single pathname results and the command expects only one,
94848 it shall be an error.

94849 The process of shell expansions in the editor shall be done as follows. The *ex* utility
94850 shall pass two arguments to the program named by the shell edit option; the first
94851 shall be **−c**, and the second shall be the string `"echo"` and the command text as a
94852 single argument. The standard output and standard error of that command shall
94853 replace the command text.

94854 **!** A character that can be appended to the command name to modify its operation,
94855 as detailed in the individual command descriptions. With the exception of the *ex*
94856 **read**, **write**, and **!** commands, the `'!'` character shall only act as a modifier if there
94857 are no <blank> characters between it and the command name.

94858 *remembered search direction*
94859 The *vi* commands **N** and **n** begin searching in a forwards or backwards direction in
94860 the edit buffer based on a remembered search direction, which is initially unset,
94861 and is set by the *ex* **global**, **v**, **s**, and **tag** commands, and the *vi* **/** and **?** commands.

94862 **Abbreviate**

94863 *Synopsis*: `ab[`*`breviate`*`] [`*`lhs rhs`*`]`

94864 If *lhs* and *rhs* are not specified, write the current list of abbreviations and do nothing more.

94865 Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable
94866 characters and <blank> characters shall not be restricted. Additional restrictions shall be
94867 implementation-defined.

94868 In both *lhs* and *rhs*, any character may be escaped with a <control>-V, in which case the character
94869 shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V shall be discarded.

94870 In open and visual text input mode, if a non-word or <ESC> character that is not escaped by a
94871 <control>-V character is entered after a word character, a check shall be made for a set of
94872 characters matching *lhs*, in the text input entered during this command. If it is found, the effect
94873 shall be as if *rhs* was entered instead of *lhs*.

94874 The set of characters that are checked is defined as follows:

94875 1. If there are no characters inserted before the word and non-word or <ESC> characters
94876 that triggered the check, the set of characters shall consist of the word character.

94877 2. If the character inserted before the word and non-word or <ESC> characters that
94878 triggered the check is a word character, the set of characters shall consist of the characters
94879 inserted immediately before the triggering characters that are word characters, plus the
94880 triggering word character.

94881 3. If the character inserted before the word and non-word or <ESC> characters that
94882 triggered the check is not a word character, the set of characters shall consist of the
94883 characters that were inserted before the triggering characters that are neither <blank>
94884 characters nor word characters, plus the triggering word character.

94885 It is unspecified whether the *lhs* argument entered for the *ex* **abbreviate** and **unabbreviate**
94886 commands is replaced in this fashion. Regardless of whether or not the replacement occurs, the
94887 effect of the command shall be as if the replacement had not occurred.

94888 *Current line*: Unchanged.

94889 *Current column*: Unchanged.

94890 **Append**

94891 *Synopsis*: **[***1addr***]** a**[**ppend**][!]**

94892 Enter *ex* text input mode; the input text shall be placed after the specified line. If line zero is
94893 specified, the text shall be placed at the beginning of the edit buffer.

94894 This command shall be affected by the **number** and **autoindent** edit options; following the
94895 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the
94896 duration of this command only.

94897 *Current line*: Set to the last input line; if no lines were input, set to the specified line, or to the first
94898 line of the edit buffer if a line of zero was specified, or zero if the edit buffer is empty.

94899 *Current column*: Set to non-<blank>.

94900 **Arguments**

94901 *Synopsis*: ar**[***gs***]**

94902 Write the current argument list, with the current argument-list entry, if any, between '[' and
94903 ']' characters.

94904 *Current line*: Unchanged.

94905 *Current column*: Unchanged.

94906 **Change**

94907 *Synopsis*: **[***2addr***]** c**[**hange**][!][***count***]**

94908 Enter *ex* text input mode; the input text shall replace the specified lines. The specified lines shall
94909 be copied into the unnamed buffer, which shall become a line mode buffer.

94910 This command shall be affected by the **number** and **autoindent** edit options; following the
94911 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the
94912 duration of this command only.

94913 *Current line*: Set to the last input line; if no lines were input, set to the line before the first
94914 address, or to the first line of the edit buffer if there are no lines preceding the first address, or to
94915 zero if the edit buffer is empty.

94916 *Current column*: Set to non-<blank>.

94917 **Change Directory**

94918 *Synopsis*: `chd[ir][!][`*`directory`*`]`
94919 `cd[!][`*`directory`*`]`

94920 Change the current working directory to *directory*.

94921 If no *directory* argument is specified, and the *HOME* environment variable is set to a non-null
94922 and non-empty value, *directory* shall default to the value named in the *HOME* environment
94923 variable. If the *HOME* environment variable is empty or is undefined, the default value of
94924 *directory* is implementation-defined.

94925 If no '!' is appended to the command name, and the edit buffer has been modified since the
94926 last complete write, and the current pathname does not begin with a '/', it shall be an error.

94927 *Current line*: Unchanged.

94928 *Current column*: Unchanged.

94929 **Copy**

94930 *Synopsis*: `[`*`2addr`*`] co[py] `*`1addr`*` [`*`flags`*`]`
94931 `[`*`2addr`*`] t `*`1addr`*` [`*`flags`*`]`

94932 Copy the specified lines after the specified destination line; line zero specifies that the lines shall
94933 be placed at the beginning of the edit buffer.

94934 *Current line*: Set to the last line copied.

94935 *Current column*: Set to non-<blank>.

94936 **Delete**

94937 *Synopsis*: `[`*`2addr`*`] d[elete][ `*`buffer`*`][`*`count`*`][`*`flags`*`]`

94938 Delete the specified lines into a buffer (defaulting to the unnamed buffer), which shall become a
94939 line-mode buffer.

94940 Flags can immediately follow the command name; see Command Line Parsing in ex (on page
94941 2846).

94942 *Current line*: Set to the line following the deleted lines, or to the last line in the edit buffer if that
94943 line is past the end of the edit buffer, or to zero if the edit buffer is empty.

94944 *Current column*: Set to non-<blank>.

94945 **Edit**

94946 *Synopsis*: `e[dit][!][+`*`command`*`][`*`file`*`]`
94947 `ex[!][+`*`command`*`][`*`file`*`]`

94948 If no '!' is appended to the command name, and the edit buffer has been modified since the
94949 last complete write, it shall be an error.

94950 If *file* is specified, replace the current contents of the edit buffer with the current contents of *file*,
94951 and set the current pathname to *file*. If *file* is not specified, replace the current contents of the
94952 edit buffer with the current contents of the file named by the current pathname. If for any reason
94953 the current contents of the file cannot be accessed, the edit buffer shall be empty.

94954 The +*command* option shall be <blank>-delimited; <blank> characters within the +*command* can
94955 be escaped by preceding them with a <backslash> character. The +*command* shall be interpreted

as an *ex* command immediately after the contents of the edit buffer have been replaced and the current line and column have been set.

If the edit buffer is empty:

Current line: Set to 0.

Current column: Set to 1.

Otherwise, if executed while in *ex* command mode or if the **+***command* argument is specified:

Current line: Set to the last line of the edit buffer.

Current column: Set to non-<blank>.

Otherwise, if *file* is omitted or results in the current pathname:

Current line: Set to the first line of the edit buffer.

Current column: Set to non-<blank>.

Otherwise, if *file* is the same as the last file edited, the line and column shall be set as follows; if the file was previously edited, the line and column may be set as follows:

Current line: Set to the last value held when that file was last edited. If this value is not a valid line in the new edit buffer, set to the first line of the edit buffer.

Current column: If the current line was set to the last value held when the file was last edited, set to the last value held when the file was last edited. Otherwise, or if the last value is not a valid column in the new edit buffer, set to non-<blank>.

Otherwise:

Current line: Set to the first line of the edit buffer.

Current column: Set to non-<blank>.

File

Synopsis: `f[ile][`*`file`*`]`

If a *file* argument is specified, the alternate pathname shall be set to the current pathname, and the current pathname shall be set to *file*.

Write an informational message. If the file has a current pathname, it shall be included in this message; otherwise, the message shall indicate that there is no current pathname. If the edit buffer contains lines, the current line number and the number of lines in the edit buffer shall be included in this message; otherwise, the message shall indicate that the edit buffer is empty. If the edit buffer has been modified since the last complete write, this fact shall be included in this message. If the **readonly** edit option is set, this fact shall be included in this message. The message may contain other unspecified information.

Current line: Unchanged.

Current column: Unchanged.

Global

Synopsis: **[***2addr***]** g**[**lobal**]** */pattern/* **[***commands***]**
[*2addr***]** v */pattern/* **[***commands***]**

The optional '!' character after the **global** command shall be the same as executing the **v** command.

If *pattern* is empty (for example, "//") or not specified, the last regular expression used in the editor command shall be used as the *pattern*. The *pattern* can be delimited by <slash> characters (shown in the Synopsis), as well as any non-alphanumeric or non-<blank> other than <backslash>, <vertical-line>, <newline>, or double-quote. Within the pattern, in certain circumstances the delimiter can be used as a literal character; see Regular Expressions in ex (on page 2875).

If no lines are specified, the lines shall default to the entire file.

The **global** and **v** commands are logically two-pass operations. First, mark the lines within the specified lines for which the line excluding the terminating <newline> matches (**global**) or does not match (**v** or **global!**) the specified pattern. Second, execute the *ex* commands given by *commands*, with the current line ('.') set to each marked line. If an error occurs during this process, or the contents of the edit buffer are replaced (for example, by the *ex* **:edit** command) an error message shall be written and no more commands resulting from the execution of this command shall be processed.

Multiple *ex* commands can be specified by entering multiple commands on a single line using a <vertical-line> to delimit them, or one per line, by escaping each <newline> with a <backslash>.

If no commands are specified:

1. If in *ex* command mode, it shall be as if the **print** command were specified.
2. Otherwise, no command shall be executed.

For the **append**, **change**, and **insert** commands, the input text shall be included as part of the command, and the terminating <period> can be omitted if the command ends the list of commands. The **open** and **visual** commands can be specified as one of the commands, in which case each marked line shall cause the editor to enter open or visual mode. If open or visual mode is exited using the *vi* **Q** command, the current line shall be set to the next marked line, and open or visual mode reentered, until the list of marked lines is exhausted.

The **global**, **v**, and **undo** commands cannot be used in *commands*. Marked lines may be deleted by commands executed for lines occurring earlier in the file than the marked lines. In this case, no commands shall be executed for the deleted lines.

If the remembered search direction is not set, the **global** and **v** commands shall set it to forward.

The **autoprint** and **autoindent** edit options shall be inhibited for the duration of the **g** or **v** command.

Current line: If no commands executed, set to the last marked line. Otherwise, as specified for the executed *ex* commands.

Current column: If no commands are executed, set to non-<blank>; otherwise, as specified for the individual *ex* commands.

95030 **Insert**

95031 *Synopsis*: **[***1addr***]** i**[**nsert**][!]**

95032 Enter *ex* text input mode; the input text shall be placed before the specified line. If the line is zero
95033 or 1, the text shall be placed at the beginning of the edit buffer.

95034 This command shall be affected by the **number** and **autoindent** edit options; following the
95035 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the
95036 duration of this command only.

95037 *Current line*: Set to the last input line; if no lines were input, set to the line before the specified
95038 line, or to the first line of the edit buffer if there are no lines preceding the specified line, or zero
95039 if the edit buffer is empty.

95040 *Current column*: Set to non-<blank>.

95041 **Join**

95042 *Synopsis*: **[***2addr***]** j**[**oin**][!][***count***][***flags***]**

95043 If *count* is specified:

95044 If no address was specified, the **join** command shall behave as if *2addr* were the current
95045 line and the current line plus *count* (.,. + *count*).

95046 If one address was specified, the **join** command shall behave as if *2addr* were the specified
95047 address and the specified address plus *count* (*addr*,*addr* + *count*).

95048 If two addresses were specified, the **join** command shall behave as if an additional
95049 address, equal to the last address plus *count* −1 (*addr1*,*addr2*,*addr2* + *count* −1), was
95050 specified.

95051 If this would result in a second address greater than the last line of the edit buffer, it shall
95052 be corrected to be equal to the last line of the edit buffer.

95053 If no *count* is specified:

95054 If no address was specified, the **join** command shall behave as if *2addr* were the current
95055 line and the next line (.,. +1).

95056 If one address was specified, the **join** command shall behave as if *2addr* were the specified
95057 address and the next line (*addr*,*addr* +1).

95058 Join the text from the specified lines together into a single line, which shall replace the specified
95059 lines.

95060 If a '!' character is appended to the command name, the **join** shall be without modification of
95061 any line, independent of the current locale.

95062 Otherwise, in the POSIX locale, set the current line to the first of the specified lines, and then, for
95063 each subsequent line, proceed as follows:

95064 1. Discard leading <space> characters from the line to be joined.

95065 2. If the line to be joined is now empty, delete it, and skip steps 3 through 5.

95066 3. If the current line ends in a <blank>, or the first character of the line to be joined is a ')'
95067 character, join the lines without further modification.

4. If the last character of the current line is a `'.'`, join the lines with two <space> characters between them.
5. Otherwise, join the lines with a single <space> between them.

Current line: Set to the first line specified.

Current column: Set to non-<blank>.

List

Synopsis: `[`*2addr*`] l[ist][`*count*`][`*flags*`]`

This command shall be equivalent to the *ex* command:

`[`*2addr*`] p[rint][`*count*`] l[`*flags*`]`

See Print (on page 2864).

Map

Synopsis: `map[!][`*lhs rhs*`]`

If *lhs* and *rhs* are not specified:

1. If `'!'` is specified, write the current list of text input mode maps.
2. Otherwise, write the current list of command mode maps.
3. Do nothing more.

Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable characters and <blank> characters shall not be restricted. Additional restrictions shall be implementation-defined. In both *lhs* and *rhs*, any character can be escaped with a <control>-V, in which case the character shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V shall be discarded.

If the character `'!'` is appended to the **map** command name, the mapping shall be effective during open or visual text input mode rather than **open** or **visual** command mode. This allows *lhs* to have two different **map** definitions at the same time: one for command mode and one for text input mode.

For command mode mappings:

> When the *lhs* is entered as any part of a *vi* command in open or visual mode (but not as part of the arguments to the command), the action shall be as if the corresponding *rhs* had been entered.
>
> If any character in the command, other than the first, is escaped using a <control>-V character, that character shall not be part of a match to an *lhs*.
>
> It is unspecified whether implementations shall support **map** commands where the *lhs* is more than a single character in length, where the first character of the *lhs* is printable.
>
> If *lhs* contains more than one character and the first character is `'#'`, followed by a sequence of digits corresponding to a numbered function key, then when this function key is typed it shall be mapped to *rhs*. Characters other than digits following a `'#'` character also represent the function key named by the characters in the *lhs* following the `'#'` and may be mapped to *rhs*. It is unspecified how function keys are named or what function keys are supported.

95107 For text input mode mappings:

95108 When the *lhs* is entered as any part of text entered in open or visual text input modes, the
95109 action shall be as if the corresponding *rhs* had been entered.

95110 If any character in the input text is escaped using a <control>-V character, that character
95111 shall not be part of a match to an *lhs*.

95112 It is unspecified whether the *lhs* text entered for subsequent **map** or **unmap** commands is
95113 replaced with the *rhs* text for the purposes of the screen display; regardless of whether or
95114 not the display appears as if the corresponding *rhs* text was entered, the effect of the
95115 command shall be as if the *lhs* text was entered.

95116 If only part of the *lhs* is entered, it is unspecified how long the editor will wait for additional,
95117 possibly matching characters before treating the already entered characters as not matching the
95118 *lhs*.

95119 The *rhs* characters shall themselves be subject to remapping, unless otherwise specified by the
95120 **remap** edit option, except that if the characters in *lhs* occur as prefix characters in *rhs*, those
95121 characters shall not be remapped.

95122 On block-mode terminals, the mapping need not occur immediately (for example, it may occur
95123 after the terminal transmits a group of characters to the system), but it shall achieve the same
95124 results as if it occurred immediately.

95125 *Current line*: Unchanged.

95126 *Current column*: Unchanged.

95127 **Mark**

95128 *Synopsis*: `[`*laddr*`]` `ma[rk]` *character*
95129 `[`*laddr*`]` `k` *character*

95130 Implementations shall support *character* values of a single lowercase letter of the POSIX locale
95131 and the backquote and single-quote characters; support of other characters is implementation-
95132 defined.

95133 If executing the *vi* **m** command, set the specified mark to the current line and 1-based numbered
95134 character referenced by the current column, if any; otherwise, column position 1.

95135 Otherwise, set the specified mark to the specified line and 1-based numbered first non-<blank>
95136 non-<newline> in the line, if any; otherwise, the last non-<newline> in the line, if any;
95137 otherwise, column position 1.

95138 The mark shall remain associated with the line until the mark is reset or the line is deleted. If a
95139 deleted line is restored by a subsequent **undo** command, any marks previously associated with
95140 the line, which have not been reset, shall be restored as well. Any use of a mark not associated
95141 with a current line in the edit buffer shall be an error.

95142 The marks ` and ' shall be set as described previously, immediately before the following events
95143 occur in the editor:

95144 1. The use of `'$'` as an *ex* address

95145 2. The use of a positive decimal number as an *ex* address

95146 3. The use of a search command as an *ex* address

4. The use of a mark reference as an *ex* address
5. The use of the following open and visual mode commands: <control>-], %, **(**, **)**, [,], {, }
6. The use of the following open and visual mode commands: **'**, **G**, **H**, **L**, **M**, **z** if the current line will change as a result of the command
7. The use of the open and visual mode commands: **/**, **?**, **N**, **`**, **n** if the current line or column will change as a result of the command
8. The use of the *ex* mode commands: **z**, **undo**, **global**, **v**

For rules 1., 2., 3., and 4., the ` and ' marks shall not be set if the *ex* command is parsed as specified by rule 6.a. in Command Line Parsing in ex (on page 2846).

For rules 5., 6., and 7., the ` and ' marks shall not be set if the commands are used as motion commands in open and visual mode.

For rules 1., 2., 3., 4., 5., 6., 7., and 8., the ` and ' marks shall not be set if the command fails.

The ` and ' marks shall be set as described previously, each time the contents of the edit buffer are replaced (including the editing of the initial buffer), if in open or visual mode, or if in **ex** mode and the edit buffer is not empty, before any commands or movements (including commands or movements specified by the **−c** or **−t** options or the **+***command* argument) are executed on the edit buffer. If in open or visual mode, the marks shall be set as if executing the *vi* **m** command; otherwise, as if executing the *ex* **mark** command.

When changing from **ex** mode to open or visual mode, if the ` and ' marks are not already set, the ` and ' marks shall be set as described previously.

Current line: Unchanged.

Current column: Unchanged.

Move

Synopsis: **[***2addr***]** m**[**ove**]** *1addr* **[***flags***]**

Move the specified lines after the specified destination line. A destination of line zero specifies that the lines shall be placed at the beginning of the edit buffer. It shall be an error if the destination line is within the range of lines to be moved.

Current line: Set to the last of the moved lines.

Current column: Set to non-<blank>.

Next

Synopsis: n**[**ext**]** **[**!**]** **[**+*command***]** **[***file ...***]**

If no '!' is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error, unless the file is successfully written as specified by the **autowrite** option.

If one or more files is specified:

1. Set the argument list to the specified filenames.
2. Set the current argument list reference to be the first entry in the argument list.

95184 3. Set the current pathname to the first filename specified.

95185 Otherwise:

95186 1. It shall be an error if there are no more filenames in the argument list after the filename
95187 currently referenced.

95188 2. Set the current pathname and the current argument list reference to the filename after the
95189 filename currently referenced in the argument list.

95190 Replace the contents of the edit buffer with the contents of the file named by the current
95191 pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be
95192 empty.

95193 This command shall be affected by the **autowrite** and **writeany** edit options.

95194 The +*command* option shall be <blank>-delimited; <blank> characters can be escaped by
95195 preceding them with a <backslash> character. The +*command* shall be interpreted as an *ex*
95196 command immediately after the contents of the edit buffer have been replaced and the current
95197 line and column have been set.

95198 *Current line*: Set as described for the **edit** command.

95199 *Current column*: Set as described for the **edit** command.

95200 **Number**

95201 *Synopsis*: `[`*2addr*`] nu[mber][`*count*`][`*flags*`]`
95202 `[`*2addr*`] #[`*count*`][`*flags*`]`

95203 These commands shall be equivalent to the *ex* command:

95204 `[`*2addr*`] p[rint][`*count*`] #[`*flags*`]`

95205 See Print (on page 2864).

95206 **Open**

95207 *Synopsis*: `[`*1addr*`] o[pen] /`*pattern*`/ [`*flags*`]`

95208 This command need not be supported on block-mode terminals or terminals with insufficient
95209 capabilities. If standard input, standard output, or standard error are not terminal devices, the
95210 results are unspecified.

95211 Enter open mode.

95212 The trailing delimiter can be omitted from *pattern* at the end of the command line. If *pattern* is
95213 empty (for example, `"//"`) or not specified, the last regular expression used in the editor shall
95214 be used as the pattern. The pattern can be delimited by <slash> characters (shown in the
95215 Synopsis), as well as any alphanumeric, or non-<blank> other than <backslash>, <vertical-line>,
95216 <newline>, or double-quote.

95217 *Current line*: Set to the specified line.

95218 *Current column*: Set to non-<blank>.

95219 **Preserve**

95220 *Synopsis*: `pre[serve]`

95221 Save the edit buffer in a form that can later be recovered by using the **–r** option or by using the
95222 *ex* **recover** command. After the file has been preserved, a mail message shall be sent to the user.
95223 This message shall be readable by invoking the *mailx* utility. The message shall contain the name
95224 of the file, the time of preservation, and an *ex* command that could be used to recover the file.
95225 Additional information may be included in the mail message.

95226 *Current line*: Unchanged.

95227 *Current column*: Unchanged.

95228 **Print**

95229 *Synopsis*: **[***2addr***]** `p[rint][`*count*`][`*flags*`]`

95230 Write the addressed lines. The behavior is unspecified if the number of columns on the display is
95231 less than the number of columns required to write any single character in the lines being written.

95232 Non-printable characters, except for the <tab>, shall be written as implementation-defined
95233 multi-character sequences.

95234 If the **#** flag is specified or the **number** edit option is set, each line shall be preceded by its line
95235 number in the following format:

95236 `"%6dΔΔ",` *<line number>*

95237 If the **l** flag is specified or the **list** edit option is set:

95238 1. The characters listed in XBD Table 5-1 (on page 113) shall be written as the corresponding
95239 escape sequence.

95240 2. Non-printable characters not in XBD Table 5-1 (on page 113) shall be written as one three-
95241 digit octal number (with a preceding <backslash>) for each byte in the character (most
95242 significant byte first).

95243 3. The end of each line shall be marked with a `'$'`, and literal `'$'` characters within the line
95244 shall be written with a preceding <backslash>.

95245 Long lines shall be folded; the length at which folding occurs is unspecified, but should be
95246 appropriate for the output terminal, considering the number of columns of the terminal.

95247 If a line is folded, and the **l** flag is not specified and the **list** edit option is not set, it is unspecified
95248 whether a multi-column character at the folding position is separated; it shall not be discarded.

95249 *Current line*: Set to the last written line.

95250 *Current column*: Unchanged if the current line is unchanged; otherwise, set to non-<blank>.

95251 **Put**

95252 *Synopsis*: **[***laddr***]** `pu[t][ `*buffer*`]`

95253 Append text from the specified buffer (by default, the unnamed buffer) to the specified line; line
95254 zero specifies that the text shall be placed at the beginning of the edit buffer. Each portion of a
95255 line in the buffer shall become a new line in the edit buffer, regardless of the mode of the buffer.

95256 *Current line*: Set to the last line entered into the edit buffer.

95257 *Current column*: Set to non-<blank>.

95258 **Quit**

95259 *Synopsis*: `q[uit][!]`

95260 If no `'!'` is appended to the command name:

95261 1. If the edit buffer has been modified since the last complete write, it shall be an error.

95262 95263 95264 2. If there are filenames in the argument list after the filename currently referenced, and the last command was not a **quit**, **wq**, **xit**, or **ZZ** (see Exit, on page 3561) command, it shall be an error.

95265 Otherwise, terminate the editing session.

95266 **Read**

95267 *Synopsis*: `[`*`1addr`*`] r[ead][!][`*`file`*`]`

95268 95269 95270 95271 95272 95273 If `'!'` is not the first non-<blank> to follow the command name, a copy of the specified file shall be appended into the edit buffer after the specified line; line zero specifies that the copy shall be placed at the beginning of the edit buffer. The number of lines and bytes read shall be written. If no *file* is named, the current pathname shall be the default. If there is no current pathname, then *file* shall become the current pathname. If there is no current pathname or *file* operand, it shall be an error. Specifying a *file* that is not of type regular shall have unspecified results.

95274 95275 Otherwise, if *file* is preceded by `'!'`, the rest of the line after the `'!'` shall have `'%'`, `'#'`, and `'!'` characters expanded as described in Command Line Parsing in ex (on page 2846).

95276 95277 95278 95279 95280 The *ex* utility shall then pass two arguments to the program named by the shell edit option; the first shall be **−c** and the second shall be the expanded arguments to the **read** command as a single argument. The standard input of the program shall be set to the standard input of the *ex* program when it was invoked. The standard error and standard output of the program shall be appended into the edit buffer after the specified line.

95281 95282 95283 95284 Each line in the copied file or program output (as delimited by <newline> characters or the end of the file or output if it is not immediately preceded by a <newline>), shall be a separate line in the edit buffer. Any occurrences of <carriage-return> and <newline> pairs in the output shall be treated as single <newline> characters.

95285 95286 The special meaning of the `'!'` following the **read** command can be overridden by escaping it with a <backslash> character.

95287 95288 95289 *Current line*: If no lines are added to the edit buffer, unchanged. Otherwise, if in open or visual mode, set to the first line entered into the edit buffer. Otherwise, set to the last line entered into the edit buffer.

95290 *Current column*: Set to non-<blank>.

95291 **Recover**

95292 *Synopsis*: `rec[over][!]` *`file`*

95293 95294 If no `'!'` is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error.

95295 95296 If no *file* operand is specified, then the current pathname shall be used. If there is no current pathname or *file* operand, it shall be an error.

95297 95298 If no recovery information has previously been saved about *file*, the **recover** command shall behave identically to the **edit** command, and an informational message to this effect shall be

95299 written.

95300 Otherwise, set the current pathname to *file*, and replace the current contents of the edit buffer
95301 with the recovered contents of *file*. If there are multiple instances of the file to be recovered, the
95302 one most recently saved shall be recovered, and an informational message that there are
95303 previous versions of the file that can be recovered shall be written. The editor shall behave as if
95304 the contents of the edit buffer have already been modified.

95305 *Current file*: Set as described for the **edit** command.

95306 *Current column*: Set as described for the **edit** command.

95307 **Rewind**

95308 *Synopsis*: `rew[ind][!]`

95309 If no `'!'` is appended to the command name, and the edit buffer has been modified since the
95310 last complete write, it shall be an error, unless the file is successfully written as specified by the
95311 **autowrite** option.

95312 If the argument list is empty, it shall be an error.

95313 The current argument list reference and the current pathname shall be set to the first filename in
95314 the argument list.

95315 Replace the contents of the edit buffer with the contents of the file named by the current
95316 pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be
95317 empty.

95318 This command shall be affected by the **autowrite** and **writeany** edit options.

95319 *Current line*: Set as described for the **edit** command.

95320 *Current column*: Set as described for the **edit** command.

95321 **Set**

95322 *Synopsis*: `se[t][` *option*`[=[`*value*`]] ...][` no*option* `...][` *option*`? ...][ all]`

95323 When no arguments are specified, write the value of the **term** edit option and those options
95324 whose values have been changed from the default settings; when the argument *all* is specified,
95325 write all of the option values.

95326 Giving an option name followed by the character `'?'` shall cause the current value of that
95327 option to be written. The `'?'` can be separated from the option name by zero or more <blank>
95328 characters. The `'?'` shall be necessary only for Boolean valued options. Boolean options can be
95329 given values by the form **set** *option* to turn them on or **set no***option* to turn them off; string and
95330 numeric options can be assigned by the form **set** *option*=*value*. Any <blank> characters in strings
95331 can be included as is by preceding each <blank> with an escaping <backslash>. More than one
95332 option can be set or listed by a single set command by specifying multiple arguments, each
95333 separated from the next by one or more <blank> characters. Arguments can appear in any order
95334 and shall be processed in the specified order.

95335 See Edit Options in ex (on page 2877) for details about specific options.

95336 *Current line*: Unchanged.

95337 *Current column*: Unchanged.

95338 **Shell**

95339 *Synopsis*: `sh[ell]`

95340 Invoke the program named in the **shell** edit option with the single argument **–i** (interactive
95341 mode). Editing shall be resumed when the program exits.

95342 *Current line*: Unchanged.

95343 *Current column*: Unchanged.

95344 **Source**

95345 *Synopsis*: `so[urce]` *file*

95346 Read and execute *ex* commands from *file*. Lines in the file that are blank lines shall be ignored.

95347 *Current line*: As specified for the individual *ex* commands.

95348 *Current column*: As specified for the individual *ex* commands.

95349 **Substitute**

95350 *Synopsis*: `[`*2addr*`] s[ubstitute][/`*pattern*`/`*repl*`/][`*options*`][`*count*`][`*flags*`]`
95351 `[`*2addr*`] &[`*options*`][`*count*`][`*flags*`]`
95352 `[`*2addr*`] ~[`*options*`][`*count*`][`*flags*`]`

95353 Replace the first instance of the pattern *pattern* by the string *repl* on each specified line. (See
95354 Regular Expressions in ex (on page 2875) and Replacement Strings in ex (on page 2876).) Any
95355 non-alphabetic, non-<blank> delimiter other than <backslash>, `'|'`, <newline>, or double-
95356 quote can be used instead of `'/'`. Within the pattern, in certain circumstances the delimiter can
95357 be used as a literal character; see Regular Expressions in ex (on page 2875). Within the
95358 replacement, the delimiter shall not terminate the replacement if it is the second character of an
95359 escape sequence (see XBD Section 9.1, on page 179) and the escaped delimiter shall be treated as
95360 that literal character in the replacement (losing any special meaning it would have had if it was
95361 not used as the delimiter and was not escaped). It shall be an error if the substitution fails on
95362 every addressed line.

95363 The trailing delimiter can be omitted from *pattern* or from *repl* at the end of the command line. If
95364 both *pattern* and *repl* are not specified or are empty (for example, `"//"`), the last **s** command
95365 shall be repeated. If only *pattern* is not specified or is empty, the last regular expression used in
95366 the editor shall be used as the pattern. If only *repl* is not specified or is empty, the pattern shall be
95367 replaced by nothing. If the entire replacement pattern is `'%'`, the last replacement pattern to an
95368 **s** command shall be used.

95369 Entering a <carriage-return> in *repl* (which requires an escaping <backslash> in *ex* mode and an
95370 escaping <control>-V in open or *vi* mode) shall split the line at that point, creating a new line in
95371 the edit buffer. The <carriage-return> shall be discarded.

95372 If *options* includes the letter `'g'` (**global**), all non-overlapping instances of the pattern in the line
95373 shall be replaced.

95374 If *options* includes the letter `'c'` (**confirm**), then before each substitution the line shall be written;
95375 the written line shall reflect all previous substitutions. On the following line, <space> characters
95376 shall be written beneath the characters from the line that are before the *pattern* to be replaced,
95377 and `'^'` characters written beneath the characters included in the *pattern* to be replaced. The *ex*
95378 utility shall then wait for a response from the user. An affirmative response shall cause the
95379 substitution to be done, while any other input shall not make the substitution. An affirmative
95380 response shall consist of a line with the affirmative response (as defined by the current locale) at

the beginning of the line. This line shall be subject to editing in the same way as the *ex* command line.

If interrupted (see the ASYNCHRONOUS EVENTS section), any modifications confirmed by the user shall be preserved in the edit buffer after the interrupt.

If the remembered search direction is not set, the **s** command shall set it to forward.

In the second Synopsis, the **&** command shall repeat the previous substitution, as if the **&** command were replaced by:

`s/`*`pattern`*`/`*`repl`*`/`

where *pattern* and *repl* are as specified in the previous **s**, **&**, or **~** command.

In the third Synopsis, the **~** command shall repeat the previous substitution, as if the `'~'` were replaced by:

`s/`*`pattern`*`/`*`repl`*`/`

where *pattern* shall be the last regular expression specified to the editor, and *repl* shall be from the previous substitution (including **&** and **~**) command.

These commands shall be affected by the *LC_MESSAGES* environment variable.

Current line: Set to the last line in which a substitution occurred, or, unchanged if no substitution occurred.

Current column: Set to non-<blank>.

Suspend

Synopsis: `su[spend][!]`
`st[op][!]`

Allow control to return to the invoking process; *ex* shall suspend itself as if it had received the SIGTSTP signal. The suspension shall occur only if job control is enabled in the invoking shell (see the description of *set* **–m**).

These commands shall be affected by the **autowrite** and **writeany** edit options.

The current **susp** character (see *stty*) shall be equivalent to the **suspend** command.

Tag

Synopsis: `ta[g][!]` *`tagstring`*

The results are unspecified if the format of a tags file is not as specified by the *ctags* utility (see *ctags*) description.

The **tag** command shall search for *tagstring* in the tag files referred to by the **tag** edit option, in the order they are specified, until a reference to *tagstring* is found. Files shall be searched from beginning to end. If no reference is found, it shall be an error and an error message to this effect shall be written. If the reference is not found, or if an error occurs while processing a file referred to in the **tag** edit option, it shall be an error, and an error message shall be written at the first occurrence of such an error.

Otherwise, if the tags file contained a pattern, the pattern shall be treated as a regular expression used in the editor; for example, for the purposes of the **s** command.

If the *tagstring* is in a file with a different name than the current pathname, set the current pathname to the name of that file, and replace the contents of the edit buffer with the contents of

95421 that file. In this case, if no `'!'` is appended to the command name, and the edit buffer has been
95422 modified since the last complete write, it shall be an error, unless the file is successfully written
95423 as specified by the **autowrite** option.

95424 This command shall be affected by the **autowrite**, **tag**, **taglength**, and **writeany** edit options.

95425 *Current line*: If the tags file contained a line number, set to that line number. If the line number is
95426 larger than the last line in the edit buffer, an error message shall be written and the current line
95427 shall be set as specified for the **edit** command.

95428 If the tags file contained a pattern, set to the first occurrence of the pattern in the file. If no
95429 matching pattern is found, an error message shall be written and the current line shall be set as
95430 specified for the **edit** command.

95431 *Current column*: If the tags file contained a line-number reference and that line-number was not
95432 larger than the last line in the edit buffer, or if the tags file contained a pattern and that pattern
95433 was found, set to non-<blank>. Otherwise, set as specified for the **edit** command.

95434 **Unabbreviate**

95435 *Synopsis*: `una[bbrev]` *`lhs`*

95436 If *lhs* is not an entry in the current list of abbreviations (see Abbreviate, on page 2854), it shall be
95437 an error. Otherwise, delete *lhs* from the list of abbreviations.

95438 *Current line*: Unchanged.

95439 *Current column*: Unchanged.

95440 **Undo**

95441 *Synopsis*: `u[ndo]`

95442 Reverse the changes made by the last command that modified the contents of the edit buffer,
95443 including **undo**. For this purpose, the **global**, **v**, **open**, and **visual** commands, and commands
95444 resulting from buffer executions and mapped character expansions, are considered single
95445 commands.

95446 If no action that can be undone preceded the **undo** command, it shall be an error.

95447 If the **undo** command restores lines that were marked, the mark shall also be restored unless it
95448 was reset subsequent to the deletion of the lines.

95449 *Current line*:

95450 1. If lines are added or changed in the file, set to the first line added or changed.
95451 2. Set to the line before the first line deleted, if it exists.
95452 3. Set to 1 if the edit buffer is not empty.
95453 4. Set to zero.

95454 *Current column*: Set to non-<blank>.

95455 **Unmap**

95456 *Synopsis*: `unm[ap][!]` *lhs*

95457 If `'!'` is appended to the command name, and if *lhs* is not an entry in the list of text input mode
95458 map definitions, it shall be an error. Otherwise, delete *lhs* from the list of text input mode map
95459 definitions.

95460 If no `'!'` is appended to the command name, and if *lhs* is not an entry in the list of command
95461 mode map definitions, it shall be an error. Otherwise, delete *lhs* from the list of command mode
95462 map definitions.

95463 *Current line*: Unchanged.

95464 *Current column*: Unchanged.

95465 **Version**

95466 *Synopsis*: `ve[rsion]`

95467 Write a message containing version information for the editor. The format of the message is
95468 unspecified.

95469 *Current line*: Unchanged.

95470 *Current column*: Unchanged.

95471 **Visual**

95472 *Synopsis*: `[`*laddr*`] vi[sual][`*type*`][`*count*`][`*flags*`]`

95473 If *ex* is currently in open or visual mode, the Synopsis and behavior of the visual command shall
95474 be the same as the **edit** command, as specified by Edit (on page 2856).

95475 Otherwise, this command need not be supported on block-mode terminals or terminals with
95476 insufficient capabilities. If standard input, standard output, or standard error are not terminal
95477 devices, the results are unspecified.

95478 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in
95479 window, on page 2883). If the `'^'` type character was also specified, the **window** edit option
95480 shall be set before being used by the type character.

95481 Enter visual mode. If *type* is not specified, it shall be as if a *type* of `'+'` was specified. The *type*
95482 shall cause the following effects:

95483 + Place the beginning of the specified line at the top of the display.

95484 − Place the end of the specified line at the bottom of the display.

95485 . Place the beginning of the specified line in the middle of the display.

95486 ^ If the specified line is less than or equal to the value of the **window** edit option, set the line
95487 to 1; otherwise, decrement the line by the value of the **window** edit option minus 1. Place
95488 the beginning of this line as close to the bottom of the displayed lines as possible, while still
95489 displaying the value of the **window** edit option number of lines.

95490 *Current line*: Set to the specified line.

95491 *Current column*: Set to non-<blank>.

95492 **Write**

95493 *Synopsis*: **[***2addr***]** **w[rite][!][>>][***file***]**
95494 **[***2addr***]** **w[rite][!][***file***]**
95495 **[***2addr***]** **wq[!][>>][***file***]**

95496 If no lines are specified, the lines shall default to the entire file.

95497 The command **wq** shall be equivalent to a **write** command followed by a **quit** command; **wq!**
95498 shall be equivalent to **write!** followed by **quit**. In both cases, if the **write** command fails, the
95499 **quit** shall not be attempted.

95500 If the command name is not followed by one or more <blank> characters, or *file* is not preceded
95501 by a `'!'` character, the **write** shall be to a file.

95502 1. If the **>>** argument is specified, and the file already exists, the lines shall be appended to
95503 the file instead of replacing its contents. If the **>>** argument is specified, and the file does
95504 not already exist, it is unspecified whether the write shall proceed as if the **>>** argument
95505 had not been specified or if the write shall fail.

95506 2. If the **readonly** edit option is set (see readonly, on page 2880), the **write** shall fail.

95507 3. If *file* is specified, and is not the current pathname, and the file exists, the **write** shall fail.

95508 4. If *file* is not specified, the current pathname shall be used. If there is no current pathname,
95509 the **write** command shall fail.

95510 5. If the current pathname is used, and the current pathname has been changed by the **file**
95511 or **read** commands, and the file exists, the **write** shall fail. If the **write** is successful,
95512 subsequent **write**s shall not fail for this reason (unless the current pathname is changed
95513 again).

95514 6. If the whole edit buffer is not being written, and the file to be written exists, the **write**
95515 shall fail.

95516 For rules 1., 2., 3., and 5., the **write** can be forced by appending the character `'!'` to the
95517 command name.

95518 For rules 2., 3., and 5., the **write** can be forced by setting the **writeany** edit option.

95519 Additional, implementation-defined tests may cause the **write** to fail.

95520 If the edit buffer is empty, a file without any contents shall be written.

95521 An informational message shall be written noting the number of lines and bytes written.

95522 Otherwise, if the command is followed by one or more <blank> characters, and the file is
95523 preceded by `'!'`, the rest of the line after the `'!'` shall have `'%'`, `'#'`, and `'!'` characters
95524 expanded as described in Command Line Parsing in ex (on page 2846).

95525 The *ex* utility shall then pass two arguments to the program named by the **shell** edit option; the
95526 first shall be **−c** and the second shall be the expanded arguments to the **write** command as a
95527 single argument. The specified lines shall be written to the standard input of the command. The
95528 standard error and standard output of the program, if any, shall be written as described for the
95529 **print** command. If the last character in that output is not a <newline>, a <newline> shall be
95530 written at the end of the output.

95531 The special meaning of the `'!'` following the **write** command can be overridden by escaping it
95532 with a <backslash> character.

95533 *Current line*: Unchanged.

95534 *Current column*: Unchanged.

95535 **Write and Exit**

95536 *Synopsis*: **[***2addr***]** x**[**it**][**!**][***file***]**

95537 If the edit buffer has not been modified since the last complete **write**, **xit** shall be equivalent to
95538 the **quit** command, or if a '!' is appended to the command name, to **quit!**.

95539 Otherwise, **xit** shall be equivalent to the **wq** command, or if a '!' is appended to the command
95540 name, to **wq!**.

95541 *Current line*: Unchanged.

95542 *Current column*: Unchanged.

95543 **Yank**

95544 *Synopsis*: **[***2addr***]** ya**[**nk**][** *buffer***][***count***]**

95545 Copy the specified lines to the specified buffer (by default, the unnamed buffer), which shall
95546 become a line-mode buffer.

95547 *Current line*: Unchanged.

95548 *Current column*: Unchanged.

95549 **Adjust Window**

95550 *Synopsis*: **[***1addr***]** z**[**!**][***type* ...**][***count***][***flags***]**

95551 If no line is specified, the current line shall be the default; if *type* is omitted as well, the current
95552 line value shall first be incremented by 1. If incrementing the current line would cause it to be
95553 greater than the last line in the edit buffer, it shall be an error.

95554 If there are <blank> characters between the *type* argument and the preceding **z** command name
95555 or optional '!' character, it shall be an error.

95556 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in
95557 window, on page 2883). If *count* is omitted, it shall default to 2 times the value of the **scroll** edit
95558 option, or if **!** was specified, the number of lines in the display minus 1.

95559 If *type* is omitted, then *count* lines starting with the specified line shall be written. Otherwise,
95560 *count* lines starting with the line specified by the *type* argument shall be written.

95561 The *type* argument shall change the lines to be written. The possible values of *type* are as follows:

95562 – The specified line shall be decremented by the following value:

95563 (((number of '−' characters) x *count*) −1)

95564 If the calculation would result in a number less than 1, it shall be an error. Write lines from
95565 the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit
95566 buffer has been written.

95567 + The specified line shall be incremented by the following value:

95568 (((number of '+' characters) −1) x *count*) +1

95569 If the calculation would result in a number greater than the last line in the edit buffer, it
95570 shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count*
95571 lines or the last line in the edit buffer has been written.

95572 95573 **=,.** If more than a single `'.'` or `'='` is specified, it shall be an error. The following steps shall be taken:

95574 1. If *count* is zero, nothing shall be written.

95575 95576 2. Write as many of the *N* lines before the current line in the edit buffer as exist. If *count* or `'!'` was specified, *N* shall be:

95577 `(count -1) /2`

95578 Otherwise, *N* shall be:

95579 `(count -3) /2`

95580 If *N* is a number less than 3, no lines shall be written.

95581 95582 3. If `'='` was specified as the type character, write a line consisting of the smaller of the number of columns in the display divided by two, or 40 `'-'` characters.

95583 4. Write the current line.

95584 5. Repeat step 3.

95585 95586 95587 6. Write as many of the *N* lines after the current line in the edit buffer as exist. *N* shall be defined as in step 2. If *N* is a number less than 3, no lines shall be written. If *count* is less than 3, no lines shall be written.

95588 **^** The specified line shall be decremented by the following value:

95589 `(((number of '^' characters) +1) x count) -1`

95590 95591 95592 If the calculation would result in a number less than 1, it shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit buffer has been written.

95593 95594 *Current line*: Set to the last line written, unless the type is **=**, in which case, set to the specified line.

95595 *Current column*: Set to non-<blank>.

95596 **Escape**

95597 95598 *Synopsis*: `!command`
`[2addr] !command`

95599 95600 95601 The contents of the line after the `'!'` shall have `'%'`, `'#'`, and `'!'` characters expanded as described in Command Line Parsing in ex (on page 2846). If the expansion causes the text of the line to change, it shall be redisplayed, preceded by a single `'!'` character.

95602 95603 95604 The *ex* utility shall execute the program named by the **shell** edit option. It shall pass two arguments to the program; the first shall be **-c**, and the second shall be the expanded arguments to the **!** command as a single argument.

95605 95606 95607 95608 If no lines are specified, the standard input, standard output, and standard error of the program shall be set to the standard input, standard output, and standard error of the *ex* program when it was invoked. In addition, a warning message shall be written if the edit buffer has been modified since the last complete write, and the **warn** edit option is set.

95609 95610 95611 95612 If lines are specified, they shall be passed to the program as standard input, and the standard output and standard error of the program shall replace those lines in the edit buffer. Each line in the program output (as delimited by <newline> characters or the end of the output if it is not immediately preceded by a <newline>), shall be a separate line in the edit buffer. Any

95613 occurrences of <carriage-return> and <newline> pairs in the output shall be treated as single
95614 <newline> characters. The specified lines shall be copied into the unnamed buffer before they
95615 are replaced, and the unnamed buffer shall become a line-mode buffer.

95616 If in *ex* mode, a single `'!'` character shall be written when the program completes.

95617 This command shall be affected by the **shell** and **warn** edit options. If no lines are specified, this
95618 command shall be affected by the **autowrite** and **writeany** edit options. If lines are specified, this
95619 command shall be affected by the **autoprint** edit option.

95620 *Current line*:

95621 1. If no lines are specified, unchanged.
95622 2. Otherwise, set to the last line read in, if any lines are read in.
95623 3. Otherwise, set to the line before the first line of the lines specified, if that line exists.
95624 4. Otherwise, set to the first line of the edit buffer if the edit buffer is not empty.
95625 5. Otherwise, set to zero.

95626 *Current column*: If no lines are specified, unchanged. Otherwise, set to non-<blank>.

95627 **Shift Left**

95628 *Synopsis*: **[***2addr***]** **<[<** **...][***count***][***flags***]**

95629 Shift the specified lines to the start of the line; the number of column positions to be shifted shall
95630 be the number of command characters times the value of the **shiftwidth** edit option. Only
95631 leading <blank> characters shall be deleted or changed into other <blank> characters in shifting;
95632 other characters shall not be affected.

95633 Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode
95634 buffer.

95635 This command shall be affected by the **autoprint** edit option.

95636 *Current line*: Set to the last line in the lines specified.

95637 *Current column*: Set to non-<blank>.

95638 **Shift Right**

95639 *Synopsis*: **[***2addr***]** **>[>** **...][***count***][***flags***]**

95640 Shift the specified lines away from the start of the line; the number of column positions to be
95641 shifted shall be the number of command characters times the value of the **shiftwidth** edit
95642 option. The shift shall be accomplished by adding <blank> characters as a prefix to the line or
95643 changing leading <blank> characters into other <blank> characters. Empty lines shall not be
95644 changed.

95645 Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode
95646 buffer.

95647 This command shall be affected by the **autoprint** edit option.

95648 *Current line*: Set to the last line in the lines specified.

95649 *Current column*: Set to non-<blank>.

95650 **<control>-D**

95651 *Synopsis*: `<control>-D`

95652 Write the next *n* lines, where *n* is the minimum of the values of the **scroll** edit option and the
95653 number of lines after the current line in the edit buffer. If the current line is the last line of the
95654 edit buffer it shall be an error.

95655 *Current line*: Set to the last line written.

95656 *Current column*: Set to non-<blank>.

95657 **Write Line Number**

95658 *Synopsis*: **[***1addr***]** `=` **[***flags***]**

95659 If *line* is not specified, it shall default to the last line in the edit buffer. Write the line number of
95660 the specified line.

95661 *Current line*: Unchanged.

95662 *Current column*: Unchanged.

95663 **Execute**

95664 *Synopsis*: **[***2addr***]** `@` *buffer*
95665 **[***2addr***]** `*` *buffer*

95666 If no buffer is specified or is specified as `'@'` or `'*'`, the last buffer executed shall be used. If no
95667 previous buffer has been executed, it shall be an error.

95668 For each line specified by the addresses, set the current line (`'.'`) to the specified line, and
95669 execute the contents of the named *buffer* (as they were at the time the @ command was executed)
95670 as *ex* commands. For each line of a line-mode buffer, and all but the last line of a character-mode
95671 buffer, the *ex* command parser shall behave as if the line was terminated by a <newline>.

95672 If an error occurs during this process, or a line specified by the addresses does not exist when
95673 the current line would be set to it, or more than a single line was specified by the addresses, and
95674 the contents of the edit buffer are replaced (for example, by the *ex* **:edit** command) an error
95675 message shall be written, and no more commands resulting from the execution of this command
95676 shall be processed.

95677 *Current line*: As specified for the individual *ex* commands.

95678 *Current column*: As specified for the individual *ex* commands.

95679 **Regular Expressions in ex**

95680 The *ex* utility shall support regular expressions that are a superset of the basic regular
95681 expressions described in XBD Section 9.3 (on page 181). A null regular expression (`"//"`) shall
95682 be equivalent to the last regular expression encountered.

95683 Regular expressions can be used in addresses to specify lines and, in some commands (for
95684 example, the **substitute** command), to specify portions of a line to be substituted.

95685 The start and end of a regular expression (RE) are marked by a delimiter character (although in
95686 some circumstances the end delimiter can be omitted). In addresses, the delimiter is either
95687 <slash> or <question-mark>. In commands, other characters can be used as the delimiter, as
95688 specified in the description of the command. Within the RE (as an *ex* extension to the BRE
95689 syntax), the delimiter shall not terminate the RE if it is the second character of an escape

sequence (see XBD Section 9.1, on page 179) and the escaped delimiter shall be treated as that literal character in the RE (losing any special meaning it would have had if it was not used as the delimiter and was not escaped). In addition, the delimiter character shall not terminate the RE when it appears within a bracket expression, and shall have its normal meaning in the bracket expression. For example, the command `"g%[%]%p"` is equivalent to `"g/[%]/p"`, and the command `"s-[0-9]--g"` is equivalent to `"s/[0-9]//g"`.

The following constructs can be used to enhance the basic regular expressions:

`\<` Match the beginning of a *word*. (See the definition of *word* at the beginning of Command Descriptions in ex (on page 2852).)

`\>` Match the end of a *word*.

`~` Match the replacement part of the last **substitute** command. The <tilde> (`'~'`) character can be escaped in a regular expression to become a normal character with no special meaning. The <backslash> shall be discarded.

When the editor option **magic** is not set, the only characters with special meanings shall be `'^'` at the beginning of a pattern, `'$'` at the end of a pattern, and <backslash>. The characters `'.'`, `'*'`, `'['`, and `'~'` shall be treated as ordinary characters unless preceded by a <backslash>; when preceded by a <backslash> they shall regain their special meaning, or in the case of <backslash>, be handled as a single <backslash>. <backslash> characters used to escape other characters shall be discarded.

Replacement Strings in ex

Certain characters and strings have special meaning in replacement strings when the character, or the first character of the string, is unescaped.

The character `'&'` (`'\&'` if the editor option **magic** is not set) in the replacement string shall stand for the text matched by the pattern to be replaced. The character `'~'` (`'\~'` if **magic** is not set) shall be replaced by the replacement part of the previous **substitute** command. The sequence `'\n'`, where *n* is an integer, shall be replaced by the text matched by the corresponding back-reference expression. If the corresponding back-reference expression does not match, then the characters `'\n'` shall be replaced by the empty string.

The strings `'\l'`, `'\u'`, `'\L'`, and `'\U'` can be used to modify the case of elements in the replacement string (using the `'\&'` or `"\"`digit) notation. The string `'\l'` (`'\u'`) shall cause the character that follows to be converted to lowercase (uppercase). The string `'\L'` (`'\U'`) shall cause all characters subsequent to it to be converted to lowercase (uppercase) as they are inserted by the substitution until the string `'\e'` or `'\E'`, or the end of the replacement string, is encountered.

Otherwise, any character following an unescaped <backslash> shall be treated as that literal character, and the escaping <backslash> shall be discarded.

An example of case conversion with the **s** command is as follows:

```
:p
The cat sat on the mat.
:s/\<.at\>/\u&/gp
The Cat Sat on the Mat.
:s/S\(.*\)M/S\U\1\eM/p
The Cat SAT ON THE Mat.
```


95733 **Edit Options in ex**

95734 The *ex* utility has a number of options that modify its behavior. These options have default
95735 settings, which can be changed using the **set** command.

95736 Options are Boolean unless otherwise specified.

95737 **autoindent, ai**

95738 [Default *unset*]

95739 If **autoindent** is set, each line in input mode shall be indented (using first as many <tab>
95740 characters as possible, as determined by the editor option **tabstop**, and then using <space>
95741 characters) to align with another line, as follows:

95742 1. If in open or visual mode and the text input is part of a line-oriented command (see the
95743 EXTENDED DESCRIPTION in *vi*), align to the first column.

95744 2. Otherwise, if in open or visual mode, indentation for each line shall be set as follows:

95745 a. If a line was previously inserted as part of this command, it shall be set to the
95746 indentation of the last inserted line by default, or as otherwise specified for the
95747 <control>-D character in Input Mode Commands in vi (on page 3561).

95748 b. Otherwise, it shall be set to the indentation of the previous current line, if any;
95749 otherwise, to the first column.

95750 3. For the *ex* **a**, **i**, and **c** commands, indentation for each line shall be set as follows:

95751 a. If a line was previously inserted as part of this command, it shall be set to the
95752 indentation of the last inserted line by default, or as otherwise specified for the *eof*
95753 character in Scroll (on page 2850).

95754 b. Otherwise, if the command is the *ex* **a** command, it shall be set to the line
95755 appended after, if any; otherwise to the first column.

95756 c. Otherwise, if the command is the *ex* **i** command, it shall be set to the line inserted
95757 before, if any; otherwise to the first column.

95758 d. Otherwise, if the command is the *ex* **c** command, it shall be set to the indentation of
95759 the line replaced.

95760 **autoprint, ap**

95761 [Default *set*]

95762 If **autoprint** is set, the current line shall be written after each *ex* command that modifies the
95763 contents of the current edit buffer, and after each **tag** command for which the tag search pattern
95764 was found or tag line number was valid, unless:

95765 1. The command was executed while in open or visual mode.

95766 2. The command was executed as part of a **global** or **v** command or @ buffer execution.

95767 3. The command was the form of the **read** command that reads a file into the edit buffer.

95768 4. The command was the **append**, **change**, or **insert** command.

95769 5. The command was not terminated by a <newline>.

6. The current line shall be written by a flag specified to the command; for example, **delete #** shall write the current line as specified for the flag modifier to the **delete** command, and not as specified by the **autoprint** edit option.

autowrite, aw

[Default *unset*]

If **autowrite** is set, and the edit buffer has been modified since it was last completely written to any file, the contents of the edit buffer shall be written as if the *ex* **write** command had been specified without arguments, before each command affected by the **autowrite** edit option is executed. Appending the character `'!'` to the command name of any of the *ex* commands except `'!'` shall prevent the write. If the write fails, it shall be an error and the command shall not be executed.

beautify, bf

XSI [Default *unset*]

If **beautify** is set, all non-printable characters, other than <tab>, <newline>, and <form-feed> characters, shall be discarded from text read in from files.

directory, dir

[Default *implementation-defined*]

The value of this option specifies the directory in which the editor buffer is to be placed. If this directory is not writable by the user, the editor shall quit.

edcompatible, ed

[Default *unset*]

Causes the presence of **g** and **c** suffixes on substitute commands to be remembered, and toggled by repeating the suffixes.

errorbells, eb

[Default *unset*]

If the editor is in *ex* mode, and the terminal does not support a standout mode (such as inverse video), and **errorbells** is set, error messages shall be preceded by alerting the terminal.

exrc

[Default *unset*]

If **exrc** is set, *ex* shall access any **.exrc** file in the current directory, as described in Initialization in ex and vi (on page 2842). If **exrc** is not set, *ex* shall ignore any **.exrc** file in the current directory during initialization, unless the current directory is that named by the *HOME* environment variable.

95803 **ignorecase, ic**

95804 [Default *unset*]

95805 If **ignorecase** is set, characters that have uppercase and lowercase representations shall have
95806 those representations considered as equivalent for purposes of regular expression comparison.

95807 The **ignorecase** edit option shall affect all remembered regular expressions; for example,
95808 unsetting the **ignorecase** edit option shall cause a subsequent *vi* **n** command to search for the
95809 last basic regular expression in a case-sensitive fashion.

95810 **list**

95811 [Default *unset*]

95812 If **list** is set, edit buffer lines written while in *ex* command mode shall be written as specified for
95813 the **print** command with the **l** flag specified. In open or visual mode, each edit buffer line shall
95814 be displayed as specified for the *ex* **print** command with the **l** flag specified. In open or visual
95815 text input mode, when the cursor does not rest on any character in the line, it shall rest on the
95816 `'$'` marking the end of the line.

95817 **magic**

95818 [Default *set*]

95819 If **magic** is set, modify the interpretation of characters in regular expressions and substitution
95820 replacement strings (see Regular Expressions in ex (on page 2875) and Replacement Strings in
95821 ex, on page 2876).

95822 **mesg**

95823 [Default *set*]

95824 If **mesg** is set, the permission for others to use the **write** or **talk** commands to write to the
95825 terminal shall be turned on while in open or visual mode. The shell-level command *mesg* **n** shall
95826 take precedence over any setting of the *ex* **mesg** option; that is, if **mesg y** was issued before the
95827 editor started (or in a shell escape), such as:

95828 `:!mesg y`

95829 the **mesg** option in *ex* shall suppress incoming messages, but the **mesg** option shall not enable
95830 incoming messages if **mesg n** was issued.

95831 **number, nu**

95832 [Default *unset*]

95833 If **number** is set, edit buffer lines written while in *ex* command mode shall be written with line
95834 numbers, in the format specified by the **print** command with the **#** flag specified. In *ex* text input
95835 mode, each line shall be preceded by the line number it will have in the file.

95836 In open or visual mode, each edit buffer line shall be displayed with a preceding line number, in
95837 the format specified by the *ex* **print** command with the **#** flag specified. This line number shall
95838 not be considered part of the line for the purposes of evaluating the current column; that is,
95839 column position 1 shall be the first column position after the format specified by the **print**
95840 command.

95841 **paragraphs, para**

95842 [Default in the POSIX locale `IPLPPPQPP LIpplpipbp`]

95843 The **paragraphs** edit option shall define additional paragraph boundaries for the open and
95844 visual mode commands. The **paragraphs** edit option can be set to a character string consisting of
95845 zero or more character pairs. It shall be an error to set it to an odd number of characters.

95846 **prompt**

95847 [Default *set*]

95848 If **prompt** is set, *ex* command mode input shall be prompted for with a <colon> (`':'`); when
95849 unset, no prompt shall be written.

95850 **readonly**

95851 [Default *see text*]

95852 If the **readonly** edit option is set, read-only mode shall be enabled (see Write, on page 2871). The
95853 **readonly** edit option shall be initialized to set if either of the following conditions are true:

95854 • The command-line option –R was specified.

95855 • Performing actions equivalent to the *access*( ) function called with the following arguments
95856 indicates that the file lacks write permission:

95857 1. The current pathname is used as the *path* argument.

95858 2. The constant **W_OK** is used as the *amode* argument.

95859 The **readonly** edit option may be initialized to set for other, implementation-defined reasons.
95860 The **readonly** edit option shall not be initialized to unset based on any special privileges of the
95861 user or process. The **readonly** edit option shall be reinitialized each time that the contents of the
95862 edit buffer are replaced (for example, by an **edit** or **next** command) unless the user has explicitly
95863 set it, in which case it shall remain set until the user explicitly unsets it. Once unset, it shall again
95864 be reinitialized each time that the contents of the edit buffer are replaced.

95865 **redraw**

95866 [Default *unset*]

95867 If **redraw** is set and the terminal is a type incapable of supporting open or visual modes, the
95868 editor shall redraw the screen when necessary in order to update its contents. (Since this is
95869 likely to require a large amount of output to the terminal, it is useful only at high transmission
95870 speeds.)

95871 **remap**

95872 [Default *set*]

95873 If **remap** is set, map translation shall allow for maps defined in terms of other maps; translation
95874 shall continue until a final product is obtained. If unset, only a one-step translation shall be
95875 done.

95876 **report**

95877 [Default 5]

95878 The value of this **report** edit option specifies what number of lines being added, copied, deleted,
95879 or modified in the edit buffer will cause an informational message to be written to the user. The
95880 following conditions shall cause an informational message. The message shall contain the
95881 number of lines added, copied, deleted, or modified, but is otherwise unspecified.

95882 • An *ex* or *vi* editor command, other than **open**, **undo**, or **visual**, that modifies at least the
95883 value of the **report** edit option number of lines, and which is not part of an *ex* **global** or **v**
95884 command, or *ex* or *vi* buffer execution, shall cause an informational message to be written.

95885 • An *ex* **yank** or *vi* **y** or **Y** command, that copies at least the value of the **report** edit option
95886 plus 1 number of lines, and which is not part of an *ex* **global** or **v** command, or *ex* or *vi*
95887 buffer execution, shall cause an informational message to be written.

95888 • An *ex* **global**, **v**, **open**, **undo**, or **visual** command or *ex* or *vi* buffer execution, that adds or
95889 deletes a total of at least the value of the **report** edit option number of lines, and which is
95890 not part of an *ex* **global** or **v** command, or *ex* or *vi* buffer execution, shall cause an
95891 informational message to be written. (For example, if 3 lines were added and 8 lines
95892 deleted during an *ex* **visual** command, 5 would be the number compared against the
95893 **report** edit option after the command completed.)

95894 **scroll, scr**

95895 [Default (number of lines in the display –1)/2]

95896 The value of the **scroll** edit option shall determine the number of lines scrolled by the *ex*
95897 <control>-D and **z** commands. For the *vi* <control>-D and <control>-U commands, it shall be the
95898 initial number of lines to scroll when no previous <control>-D or <control>-U command has
95899 been executed.

95900 **sections**

95901 [Default in the POSIX locale `NHSHH HUnhsh`]

95902 The **sections** edit option shall define additional section boundaries for the open and visual mode
95903 commands. The **sections** edit option can be set to a character string consisting of zero or more
95904 character pairs; it shall be an error to set it to an odd number of characters.

95905 **shell, sh**

95906 [Default from the environment variable *SHELL*]

95907 The value of this option shall be a string. The default shall be taken from the *SHELL*
95908 environment variable. If the *SHELL* environment variable is null or empty, the *sh* (see *sh*) utility
95909 shall be the default.

95910 **shiftwidth, sw**

95911 [Default 8]

95912 The value of this option shall give the width in columns of an indentation level used during
95913 autoindentation and by the shift commands (**<** and **>**).

95914 **showmatch, sm**

95915 [Default *unset*]

95916 The functionality described for the **showmatch** edit option need not be supported on block-
95917 mode terminals or terminals with insufficient capabilities.

95918 If **showmatch** is set, in open or visual mode, when a `')'` or `'}'` is typed, if the matching `'('` or
95919 `'{'` is currently visible on the display, the matching `'('` or `'{'` shall be flagged moving the
95920 cursor to its location for an unspecified amount of time.

95921 **showmode**

95922 [Default *unset*]

95923 If **showmode** is set, in open or visual mode, the current mode that the editor is in shall be
95924 displayed on the last line of the display. Command mode and text input mode shall be
95925 differentiated; other unspecified modes and implementation-defined information may be
95926 displayed.

95927 **slowopen**

95928 [Default *unset*]

95929 If **slowopen** is set during open and visual text input modes, the editor shall not update portions
95930 of the display other than those display line columns that display the characters entered by the
95931 user (see Input Mode Commands in vi, on page 3561).

95932 **tabstop, ts**

95933 [Default 8]

95934 The value of this edit option shall specify the column boundary used by a <tab> in the display
95935 (see autoprint, ap (on page 2877) and Input Mode Commands in vi, on page 3561).

95936 **taglength, tl**

95937 [Default zero]

95938 The value of this edit option shall specify the maximum number of characters that are
95939 considered significant in the user-specified tag name and in the tag name from the tags file. If
95940 the value is zero, all characters in both tag names shall be significant.

95941 **tags**

95942 [Default *see text*]

95943 The value of this edit option shall be a string of <blank>-delimited pathnames of files used by
95944 the **tag** command. The default value is unspecified.

95945 **term**

95946 [Default from the environment variable *TERM*]

95947 The value of this edit option shall be a string. The default shall be taken from the *TERM* variable
95948 in the environment. If the *TERM* environment variable is empty or null, the default is
95949 unspecified. The editor shall use the value of this edit option to determine the type of the display
95950 device.

95951 The results are unspecified if the user changes the value of the term edit option after editor
95952 initialization.

95953 **terse**

95954 [Default *unset*]

95955 If **terse** is set, error messages may be less verbose. However, except for this caveat, error
95956 messages are unspecified. Furthermore, not all error messages need change for different settings
95957 of this option.

95958 **warn**

95959 [Default *set*]

95960 If **warn** is set, and the contents of the edit buffer have been modified since they were last
95961 completely written, the editor shall write a warning message before certain **!** commands (see
95962 Escape, on page 2873).

95963 **window**

95964 [Default *see text*]

95965 A value used in open and visual mode, by the <control>-B and <control>-F commands, and, in
95966 visual mode, to specify the number of lines displayed when the screen is repainted.

95967 If the −**w** command-line option is not specified, the default value shall be set to the value of the
95968 *LINES* environment variable. If the *LINES* environment variable is empty or null, the default
95969 shall be the number of lines in the display minus 1.

95970 Setting the **window** edit option to zero or to a value greater than the number of lines in the
95971 display minus 1 (either explicitly or based on the −**w** option or the *LINES* environment variable)
95972 shall cause the **window** edit option to be set to the number of lines in the display minus 1.

95973 The baud rate of the terminal line may change the default in an implementation-defined manner.

wrapmargin, wm

[Default 0]

If the value of this edit option is zero, it shall have no effect.

If not in the POSIX locale, the effect of this edit option is implementation-defined.

Otherwise, it shall specify a number of columns from the ending margin of the terminal.

During open and visual text input modes, for each character for which any part of the character is displayed in a column that is less than **wrapmargin** columns from the ending margin of the display line, the editor shall behave as follows:

1. If the character triggering this event is a <blank>, it, and all immediately preceding <blank> characters on the current line entered during the execution of the current text input command, shall be discarded, and the editor shall behave as if the user had entered a single <newline> instead. In addition, if the next user-entered character is a <space>, it shall be discarded as well.
2. Otherwise, if there are one or more <blank> characters on the current line immediately preceding the last group of inserted non-<blank> characters which was entered during the execution of the current text input command, the <blank> characters shall be replaced as if the user had entered a single <newline> instead.

If the **autoindent** edit option is set, and the events described in 1. or 2. are performed, any <blank> characters at or after the cursor in the current line shall be discarded.

The ending margin shall be determined by the system or overridden by the user, as described for *COLUMNS* in the ENVIRONMENT VARIABLES section and XBD Chapter 8 (on page 167).

wrapscan, ws

[Default *set*]

If **wrapscan** is set, searches (the *ex* **/** or **?** addresses, or open and visual mode **/**, **?**, **N**, and **n** commands) shall wrap around the beginning or end of the edit buffer; when unset, searches shall stop at the beginning or end of the edit buffer.

writeany, wa

[Default *unset*]

If **writeany** is set, some of the checks performed when executing the *ex* **write** commands shall be inhibited, as described in editor option **autowrite**.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

When any error is encountered and the standard input is not a terminal device file, in addition to the default requirements described in Section 1.4 (on page 2462), *ex* shall neither write the file (if one has been opened) nor return to command or text input mode.

Otherwise, when an unrecoverable error is encountered, it shall be equivalent to a SIGHUP asynchronous event.

Otherwise, when an error is encountered, the editor shall behave as specified in Command Line Parsing in ex (on page 2846).

APPLICATION USAGE

If a SIGSEGV signal is received while *ex* is saving a file, the file might not be successfully saved.

The **next** command can accept more than one file, so usage such as:

```
next `ls [abc]*`
```

is valid; it would not be valid for the **edit** or **read** commands, for example, because they expect only one file and unspecified results occur.

Unlike the *system*() function, *ex* does not pass `"--"` between the `"-c"` argument and the command string, so that programs for which **−c** takes an option-argument can be used in the **shell** edit option. Users who want to use an **escape** command to execute a utility whose name starts with `'-'` or `'+'` need to provide a pathname for that utility that does not start with either of those characters, or precede the utility name with a <blank> character.

EXAMPLES

None.

RATIONALE

The *ex*/*vi* specification is based on the historical practice found in the 4 BSD and System V implementations of *ex* and *vi*.

A *restricted editor* (both the historical *red* utility and modifications to *ex*) were considered and rejected for inclusion. Neither option provided the level of security that users might expect.

It is recognized that *ex* visual mode and related features would be difficult, if not impossible, to implement satisfactorily on a block-mode terminal, or a terminal without any form of cursor addressing; thus, it is not a mandatory requirement that such features should work on all terminals. It is the intention, however, that an *ex* implementation should provide the full set of capabilities on all terminals capable of supporting them.

Options

The **−c** replacement for **+***command* was inspired by the **−e** option of *sed*. Historically, all such commands (see **edit** and **next** as well) were executed from the last line of the edit buffer. This meant, for example, that `"+/pattern"` would fail unless the **wrapscan** option was set. POSIX.1-2024 requires conformance to historical practice. The +*command* option is no longer specified by POSIX.1-2024 but may be present in some implementations. Historically, some implementations restricted the *ex* commands that could be listed as part of the command line arguments. For consistency, POSIX.1-2024 does not permit these restrictions.

In historical implementations of the editor, the **−R** option (and the **readonly** edit option) only prevented overwriting of files; appending to files was still permitted, mapping loosely into the *csh* **noclobber** variable. Some implementations, however, have not followed this semantic, and **readonly** does not permit appending either. POSIX.1-2024 follows the latter practice, believing that it is a more obvious and intuitive meaning of **readonly**.

The **−s** option suppresses all interactive user feedback and is useful for editing scripts in batch jobs. The list of specific effects is historical practice. The terminal type ``incapable of supporting open and visual modes'' has historically been named ``dumb''.

The **−t** option was required because the *ctags* utility appears in POSIX.1-2024 and the option is available in all historical implementations of *ex*.

Historically, the *ex* and *vi* utilities accepted a **−x** option, which did encryption based on the

96058 algorithm found in the historical *crypt* utility. The **−x** option for encryption, and the associated
96059 *crypt* utility, were omitted because the algorithm used was not specifiable and the export control
96060 laws of some nations make it difficult to export cryptographic technology. In addition, it did not
96061 historically provide the level of security that users might expect.

96062 **Standard Input**

96063 An end-of-file condition is not equivalent to an end-of-file character. A common end-of-file
96064 character, <control>-D, is historically an *ex* command.

96065 There was no maximum line length in historical implementations of *ex*. Specifically, as it was
96066 parsed in chunks, the addresses had a different maximum length than the filenames. Further, the
96067 maximum line buffer size was declared as BUFSIZ, which was different lengths on different
96068 systems. This version selected the value of {LINE_MAX} to impose a reasonable restriction on
96069 portable usage of *ex* and to aid test suite writers in their development of realistic tests that
96070 exercise this limit.

96071 **Input Files**

96072 It was an explicit decision by the standard developers that a <newline> be added to any file
96073 lacking one. It was believed that this feature of *ex* and *vi* was relied on by users in order to make
96074 text files lacking a trailing <newline> more portable. It is recognized that this will require a user-
96075 specified option or extension for implementations that permit *ex* and *vi* to edit files of type other
96076 than text if such files are not otherwise identified by the system. It was agreed that the ability to
96077 edit files of arbitrary type can be useful, but it was not considered necessary to mandate that an
96078 *ex* or *vi* implementation be required to handle files other than text files.

96079 The paragraph in the INPUT FILES section, ``By default, ...'', is intended to close a long-
96080 standing security problem in *ex* and *vi*; that of the ``modeline'' or ``modelines'' edit option. This
96081 feature allows any line in the first or last five lines of the file containing the strings `"ex:"` or
96082 `"vi:"` (and, apparently, `"ei:"` or `"vx:"`) to be a line containing editor commands, and *ex*
96083 interprets all the text up to the next `':'` or <newline> as a command. Consider the
96084 consequences, for example, of an unsuspecting user using *ex* or *vi* as the editor when replying to
96085 a mail message in which a line such as:

96086 `ex:! rm -rf :`

96087 appeared in the signature lines. The standard developers believed strongly that an editor should
96088 not by default interpret any lines of a file. Vendors are strongly urged to delete this feature from
96089 their implementations of *ex* and *vi*.

96090 **Asynchronous Events**

96091 The intention of the phrase ``complete write'' is that the entire edit buffer be written to stable
96092 storage. The note regarding temporary files is intended for implementations that use temporary
96093 files to back edit buffers unnamed by the user.

96094 Historically, SIGQUIT was ignored by *ex*, but was the equivalent of the **Q** command in visual
96095 mode; that is, it exited visual mode and entered *ex* mode. POSIX.1-2024 permits, but does not
96096 require, this behavior. Historically, SIGINT was often used by *vi* users to terminate text input
96097 mode (<control>-C is often easier to enter than <ESC>). Some implementations of *vi* alerted the
96098 terminal on this event, and some did not. POSIX.1-2024 requires that SIGINT behave identically
96099 to <ESC>, and that the terminal not be alerted.

96100 Historically, suspending the *ex* editor during text input mode was similar to SIGINT, as
96101 completed lines were retained, but any partial line discarded, and the editor returned to

command mode. POSIX.1-2024 is silent on this issue; implementations are encouraged to follow historical practice, where possible.

Historically, the *vi* editor did not treat SIGTSTP as an asynchronous event, and it was therefore impossible to suspend the editor in visual text input mode. There are two major reasons for this. The first is that SIGTSTP is a broadcast signal on UNIX systems, and the chain of events where the shell *exec*s an application that then *exec*s *vi* usually caused confusion for the terminal state if SIGTSTP was delivered to the process group in the default manner. The second was that most implementations of the UNIX *curses* package did not handle SIGTSTP safely, and the receipt of SIGTSTP at the wrong time would cause them to crash. POSIX.1-2024 is silent on this issue; implementations are encouraged to treat suspension as an asynchronous event if possible.

Historically, modifications to the edit buffer made before SIGINT interrupted an operation were retained; that is, anywhere from zero to all of the lines to be modified might have been modified by the time the SIGINT arrived. These changes were not discarded by the arrival of SIGINT. POSIX.1-2024 permits this behavior, noting that the **undo** command is required to be able to undo these partially completed commands.

The action taken for signals other than SIGINT, SIGCONT, SIGHUP, and SIGTERM is unspecified because some implementations attempt to save the edit buffer in a useful state when other signals are received.

Standard Error

For *ex*/*vi*, diagnostic messages are those messages reported as a result of a failed attempt to invoke *ex* or *vi*, such as invalid options or insufficient resources, or an abnormal termination condition. Diagnostic messages should not be confused with the error messages generated by inappropriate or illegal user commands.

Initialization in ex and vi

If an *ex* command (other than **cd**, **chdir**, or **source**) has a filename argument, one or both of the alternate and current pathnames will be set. Informally, they are set as follows:

1. If the *ex* command is one that replaces the contents of the edit buffer, and it succeeds, the current pathname will be set to the filename argument (the first filename argument in the case of the **next** command) and the alternate pathname will be set to the previous current pathname, if there was one.
2. In the case of the file read/write forms of the **read** and **write** commands, if there is no current pathname, the current pathname will be set to the filename argument.
3. Otherwise, the alternate pathname will be set to the filename argument.

For example, **:edit foo** and **:recover foo**, when successful, set the current pathname, and, if there was a previous current pathname, the alternate pathname. The commands **:write**, **!command**, and **:edit** set neither the current or alternate pathnames. If the **:edit foo** command were to fail for some reason, the alternate pathname would be set. The **read** and **write** commands set the alternate pathname to their *file* argument, unless the current pathname is not set, in which case they set the current pathname to their *file* arguments. The alternate pathname was not historically set by the **:source** command. POSIX.1-2024 requires conformance to historical practice. Implementations adding commands that take filenames as arguments are encouraged to set the alternate pathname as described here.

Historically, *ex* and *vi* read the **.exrc** file in the *$HOME* directory twice, if the editor was executed in the *$HOME* directory. POSIX.1-2024 prohibits this behavior.

Historically, the 4 BSD *ex* and *vi* read the *$HOME* and local **.exrc** files if they were owned by the real ID of the user, or the **sourceany** option was set, regardless of other considerations. This was a security problem because it is possible to put normal UNIX system commands inside a **.exrc** file. POSIX.1-2024 does not specify the **sourceany** option, and historical implementations are encouraged to delete it.

The **.exrc** files must be owned by the real ID of the user, and not writable by anyone other than the owner. The appropriate privileges exception is intended to permit users to acquire special privileges, but continue to use the **.exrc** files in their home directories.

System V Release 3.2 and later *vi* implementations added the option **[no]exrc**. The behavior is that local **.exrc** files are read-only if the **exrc** option is set. The default for the **exrc** option was off, so by default, local **.exrc** files were not read. The problem this was intended to solve was that System V permitted users to give away files, so there is no possible ownership or writeability test to ensure that the file is safe. This is still a security problem on systems where users can give away files, but there is nothing additional that POSIX.1-2024 can do. The implementation-defined exception is intended to permit groups to have local **.exrc** files that are shared by users, by creating pseudo-users to own the shared files.

POSIX.1-2024 does not mention system-wide *ex* and *vi* start-up files. While they exist in several implementations of *ex* and *vi*, they are not present in any implementations considered historical practice by POSIX.1-2024. Implementations that have such files should use them only if they are owned by the real user ID or an appropriate user (for example, root on UNIX systems) and if they are not writable by any user other than their owner. System-wide start-up files should be read before the *EXINIT* variable, **$HOME/.exrc**, or local **.exrc** files are evaluated.

Historically, any *ex* command could be entered in the *EXINIT* variable or the **.exrc** file, although ones requiring that the edit buffer already contain lines of text generally caused historical implementations of the editor to drop **core**. POSIX.1-2024 requires that any *ex* command be permitted in the *EXINIT* variable and **.exrc** files, for simplicity of specification and consistency, although many of them will obviously fail under many circumstances.

The initialization of the contents of the edit buffer uses the phrase ``the effect shall be'' with regard to various *ex* commands. The intent of this phrase is that edit buffer contents loaded during the initialization phase not be lost; that is, loading the edit buffer should fail if the **.exrc** file read in the contents of a file and did not subsequently write the edit buffer. An additional intent of this phrase is to specify that the initial current line and column is set as specified for the individual *ex* commands.

Historically, the **−t** option behaved as if the tag search were a **+***command*; that is, it was executed from the last line of the file specified by the tag. This resulted in the search failing if the pattern was a forward search pattern and the **wrapscan** edit option was not set. POSIX.1-2024 does not permit this behavior, requiring that the search for the tag pattern be performed on the entire file, and, if not found, that the current line be set to a more reasonable location in the file.

Historically, the empty edit buffer presented for editing when a file was not specified by the user was unnamed. This is permitted by POSIX.1-2024; however, implementations are encouraged to provide users a temporary filename for this buffer because it permits them the use of *ex* commands that use the current pathname during temporary edit sessions.

Historically, the file specified using the **−t** option was not part of the current argument list. This practice is permitted by POSIX.1-2024; however, implementations are encouraged to include its name in the current argument list for consistency.

Historically, the **−c** command was generally not executed until a file that already exists was edited. POSIX.1-2024 requires conformance to this historical practice. Commands that could

cause the −**c** command to be executed include the *ex* commands **edit**, **next**, **recover**, **rewind**, and **tag**, and the *vi* commands <control>-^ and <control>-]. Historically, reading a file into an edit buffer did not cause the −**c** command to be executed (even though it might set the current pathname) with the exception that it did cause the −**c** command to be executed if: the editor was in *ex* mode, the edit buffer had no current pathname, the edit buffer was empty, and no read commands had yet been attempted. For consistency and simplicity of specification, POSIX.1-2024 does not permit this behavior.

Historically, the −**r** option was the same as a normal edit session if there was no recovery information available for the file. This allowed users to enter:

```
vi -r *.c
```

and recover whatever files were recoverable. In some implementations, recovery was attempted only on the first file named, and the file was not entered into the argument list; in others, recovery was attempted for each file named. In addition, some historical implementations ignored −**r** if −**t** was specified or did not support command line *file* arguments with the −**t** option. For consistency and simplicity of specification, POSIX.1-2024 disallows these special cases, and requires that recovery be attempted the first time each file is edited.

Historically, *vi* initialized the ` and ' marks, but *ex* did not. This meant that if the first command in *ex* mode was **visual** or if an *ex* command was executed first (for example, *vi* +10 *file*), *vi* was entered without the marks being initialized. Because the standard developers believed the marks to be generally useful, and for consistency and simplicity of specification, POSIX.1-2024 requires that they always be initialized if in open or visual mode, or if in *ex* mode and the edit buffer is not empty. Not initializing it in *ex* mode if the edit buffer is empty is historical practice; however, it has always been possible to set (and use) marks in empty edit buffers in open and visual mode edit sessions.

Addressing

Historically, *ex* and *vi* accepted the additional addressing forms `'\/'` and `'\?'`. They were equivalent to `"//"` and `"??"`, respectively. They are not required by POSIX.1-2024, mostly because nobody can remember whether they ever did anything different historically.

Historically, *ex* and *vi* permitted an address of zero for several commands, and permitted the % address in empty files for others. For consistency, POSIX.1-2024 requires support for the former in the few commands where it makes sense, and disallows it otherwise. In addition, because POSIX.1-2024 requires that % be logically equivalent to `"1,$"`, it is also supported where it makes sense and disallowed otherwise.

Historically, the % address could not be followed by further addresses. For consistency and simplicity of specification, POSIX.1-2024 requires that additional addresses be supported.

All of the following are valid *addresses*:

`+++`	Three lines after the current line.
`/`*re*`/-`	One line before the next occurrence of *re*.
`-2`	Two lines before the current line.
`3 ---- 2`	Line one (note intermediate negative address).
`1 2 3`	Line six.

Any number of addresses can be provided to commands taking addresses; for example, `"1,2,3,4,5p"` prints lines 4 and 5, because two is the greatest valid number of addresses accepted by the **print** command. This, in combination with the <semicolon> delimiter, permits

96237 users to create commands based on ordered patterns in the file. For example, the command
96238 **3;/foo/;+2print** will display the first line after line 3 that contains the pattern *foo*, plus the next
96239 two lines. Note that the address **3;** must be evaluated before being discarded because the search
96240 origin for the **/foo/** command depends on this.

96241 Historically, values could be added to addresses by including them after one or more <blank>
96242 characters; for example, **3 − 5p** wrote the seventh line of the file, and **/foo/ 5** was the same as
96243 **/foo/+5**. However, only absolute values could be added; for example, **5 /foo/** was an error.
96244 POSIX.1-2024 requires conformance to historical practice. Address offsets are separately
96245 specified from addresses because they could historically be provided to visual mode search
96246 commands.

96247 Historically, any missing addresses defaulted to the current line. This was true for leading and
96248 trailing <comma>-delimited addresses, and for trailing <semicolon>-delimited addresses. For
96249 consistency, POSIX.1-2024 requires it for leading <semicolon> addresses as well.

96250 Historically, *ex* and *vi* accepted the `'^'` character as both an address and as a flag offset for
96251 commands. In both cases it was identical to the `'−'` character. POSIX.1-2024 does not require or
96252 prohibit this behavior.

96253 Historically, the enhancements to basic regular expressions could be used in addressing; for
96254 example, `'~'`, `'\<'`, and `'\>'`. POSIX.1-2024 requires conformance to historical practice; that
96255 is, that regular expression usage be consistent, and that regular expression enhancements be
96256 supported wherever regular expressions are used.

96257 **Command Line Parsing in ex**

96258 Historical *ex* command parsing was even more complex than that described here. POSIX.1-2024
96259 requires the subset of the command parsing that the standard developers believed was
96260 documented and that users could reasonably be expected to use in a portable fashion, and that
96261 was historically consistent between implementations. (The discarded functionality is obscure, at
96262 best.) Historical implementations will require changes in order to comply with POSIX.1-2024;
96263 however, users are not expected to notice any of these changes. Most of the complexity in *ex*
96264 parsing is to handle three special termination cases:

96265 1. The **!**, **global**, **v**, and the filter versions of the **read** and **write** commands are delimited by
96266 <newline> characters (they can contain <vertical-line> characters that are usually shell
96267 pipes).

96268 2. The **ex**, **edit**, **next**, and **visual** in open and visual mode commands all take *ex* commands,
96269 optionally containing <vertical-line> characters, as their first arguments.

96270 3. The **s** command takes a regular expression as its first argument, and uses the delimiting
96271 characters to delimit the command.

96272 Historically, <vertical-line> characters in the **+***command* argument of the **ex**, **edit**, **next**, **vi**, and
96273 **visual** commands, and in the *pattern* and *replacement* parts of the **s** command, did not delimit the
96274 command, and in the filter cases for **read** and **write**, and the **!**, **global**, and **v** commands, they did
96275 not delimit the command at all. For example, the following commands are all valid:

```
96276 :edit +25 | s/abc/ABC/ file.c
96277 :s/ | /PIPE/
96278 :read !spell % | columnate
96279 :global/pattern/p | l
96280 :s/a/b/ | s/c/d | set
```

96281 Historically, empty or <blank> filled lines in **.exrc** files and **source**d files (as well as *EXINIT*

variables and *ex* command scripts) were treated as default commands; that is, **print** commands. POSIX.1-2024 specifically requires that they be ignored when encountered in **.exrc** and **source**d files to eliminate a common source of new user error.

Historically, *ex* commands with multiple adjacent (or <blank>-separated) vertical lines were handled oddly when executed from *ex* mode. For example, the command **|||** <carriage-return>, when the cursor was on line 1, displayed lines 2, 3, and 5 of the file. In addition, the command **|** would only display the line after the next line, instead of the next two lines. The former worked more logically when executed from *vi* mode, and displayed lines 2, 3, and 4. POSIX.1-2024 requires the *vi* behavior; that is, a single default command and line number increment for each command separator, and trailing <newline> characters after <vertical-line> separators are discarded.

Historically, *ex* permitted a single extra <colon> as a leading command character; for example, **:g/pattern/:p** was a valid command. POSIX.1-2024 generalizes this to require that any number of leading <colon> characters be stripped.

Historically, any prefix of the **delete** command could be followed without intervening <blank> characters by a flag character because in the command **d p**, *p* is interpreted as the buffer *p*. POSIX.1-2024 requires conformance to historical practice.

Historically, the **k** command could be followed by the mark name without intervening <blank> characters. POSIX.1-2024 requires conformance to historical practice.

Historically, the **s** command could be immediately followed by flag and option characters; for example, **s/e/E/|s|sgc3p** was a valid command. However, flag characters could not stand alone; for example, the commands **sp** and **s l** would fail, while the command **sgp** and **s gl** would succeed. (Obviously, the **'#'** flag character was used as a delimiter character if it followed the command.) Another issue was that option characters had to precede flag characters even when the command was fully specified; for example, the command **s/e/E/pg** would fail, while the command **s/e/E/gp** would succeed. POSIX.1-2024 requires conformance to historical practice.

Historically, the first command name that had a prefix matching the input from the user was the executed command; for example, **ve**, **ver**, and **vers** all executed the **version** command. Commands were in a specific order, however, so that **a** matched **append**, not **abbreviate**. POSIX.1-2024 requires conformance to historical practice. The restriction on command search order for implementations with extensions is to avoid the addition of commands such that the historical prefixes would fail to work portably.

Historical implementations of *ex* and *vi* did not correctly handle multiple *ex* commands, separated by <vertical-line> characters, that entered or exited visual mode or the editor. Because implementations of *vi* exist that do not exhibit this failure mode, POSIX.1-2024 does not permit it.

The requirement that alphabetic command names consist of all following alphabetic characters up to the next non-alphabetic character means that alphabetic command names must be separated from their arguments by one or more non-alphabetic characters, normally a <blank> or **'!'** character, except as specified for the exceptions, the **delete**, **k**, and **s** commands.

Historically, the repeated execution of the *ex* default **print** commands (<control>-D, *eof*, <newline>, <carriage-return>) erased any prompting character and displayed the next lines without scrolling the terminal; that is, immediately below any previously displayed lines. This provided a cleaner presentation of the lines in the file for the user. POSIX.1-2024 does not require this behavior because it may be impossible in some situations; however, implementations are strongly encouraged to provide this semantic if possible.

Historically, it was possible to change files in the middle of a command, and have the rest of the

command executed in the new file; for example:

```
:edit +25 file.c | s/abc/ABC/ | 1
```

was a valid command, and the substitution was attempted in the newly edited file. POSIX.1-2024 requires conformance to historical practice. The following commands are examples that exercise the *ex* parser:

```
echo 'foo | bar' > file1; echo 'foo/bar' > file2;
vi
:edit +1 | s/|/PIPE/ | w file1 | e file2 | 1 | s/\//SLASH/ | wq
```

Historically, there was no protection in editor implementations to avoid *ex* **global**, **v**, **@**, or * commands changing edit buffers during execution of their associated commands. Because this would almost invariably result in catastrophic failure of the editor, and implementations exist that do exhibit these problems, POSIX.1-2024 requires that changing the edit buffer during a **global** or **v** command, or during a **@** or * command for which there will be more than a single execution, be an error. Implementations supporting multiple edit buffers simultaneously are strongly encouraged to apply the same semantics to switching between buffers as well.

The *ex* command quoting required by POSIX.1-2024 is a superset of the quoting in historical implementations of the editor. For example, it was not historically possible to escape a <blank> in a filename; for example, **:edit foo\\\ bar** would report that too many filenames had been entered for the edit command, and there was no method of escaping a <blank> in the first argument of an **edit**, **ex**, **next**, or **visual** command at all. POSIX.1-2024 extends historical practice, requiring that quoting behavior be made consistent across all *ex* commands, except for the **map**, **unmap**, **abbreviate**, and **unabbreviate** commands, which historically used <control>-V instead of <backslash> characters for quoting. For those four commands, POSIX.1-2024 requires conformance to historical practice.

Backslash quoting in *ex* is non-intuitive. <backslash>-escapes are ignored unless they escape a special character; for example, when performing *file* argument expansion, the string `"\\%"` is equivalent to `'\%'`, not `"\<`*current pathname*`>"`. This can be confusing for users because <backslash> is usually one of the characters that causes shell expansion to be performed, and therefore shell quoting rules must be taken into consideration. Generally, quoting characters are only considered if they escape a special character, and a quoting character must be provided for each layer of parsing for which the character is special. As another example, only a single <backslash> is necessary for the `'\l'` sequence in substitute replacement patterns, because the character `'l'` is not special to any parsing layer above it.

<control>-V quoting in *ex* is slightly different from backslash quoting. In the four commands where <control>-V quoting applies (**abbreviate**, **unabbreviate**, **map**, and **unmap**), any character may be escaped by a <control>-V whether it would have a special meaning or not. POSIX.1-2024 requires conformance to historical practice.

Historical implementations of the editor did not require delimiters within character classes to be escaped; for example, the command **:s/[/]//** on the string `"xxx/yyy"` would delete the `'/'` from the string. POSIX.1-2024 disallows this historical practice for consistency and because it places a large burden on implementations by requiring that knowledge of regular expressions be built into the editor parser.

Historically, quoting <newline> characters in *ex* commands was handled inconsistently. In most cases, the <newline> character always terminated the command, regardless of any preceding escape character, because <backslash> characters did not escape <newline> characters for most *ex* commands. However, some *ex* commands (for example, **s**, **map**, and **abbreviation**) permitted <newline> characters to be escaped (although in the case of **map** and **abbreviation**, <control>-V

96376 characters escaped them instead of <backslash> characters). This was true in not only the
96377 command line, but also **.exrc** and **source**d files. For example, the command:

96378 `map = foo<control-V><newline>bar`

96379 would succeed, although it was sometimes difficult to get the <control>-V and the inserted
96380 <newline> passed to the *ex* parser. For consistency and simplicity of specification, POSIX.1-2024
96381 requires that it be possible to escape <newline> characters in *ex* commands at all times, using
96382 <backslash> characters for most *ex* commands, and using <control>-V characters for the **map**
96383 and **abbreviation** commands. For example, the command **print**<newline>**list** is required to be
96384 parsed as the single command **print**<newline>**list**. While this differs from historical practice,
96385 POSIX.1-2024 developers believed it unlikely that any script or user depended on the historical
96386 behavior.

96387 Historically, an error in a command specified using the **−c** option did not cause the rest of the **−c**
96388 commands to be discarded. POSIX.1-2024 disallows this for consistency with mapped keys, the
96389 **@**, **global**, **source**, and **v** commands, the *EXINIT* environment variable, and the **.exrc** files.

96390 **Input Editing in ex**

96391 One of the common uses of the historical *ex* editor is over slow network connections. Editors that
96392 run in canonical mode can require far less traffic to and from, and far less processing on, the host
96393 machine, as well as more easily supporting block-mode terminals. For these reasons,
96394 POSIX.1-2024 requires that *ex* be implemented using canonical mode input processing, as was
96395 done historically.

96396 POSIX.1-2024 does not require the historical 4 BSD input editing characters ``word erase'' or
96397 ``literal next''. For this reason, it is unspecified how they are handled by *ex*, although they must
96398 have the required effect. Implementations that resolve them after the line has been ended using a
96399 <newline> or <control>-M character, and implementations that rely on the underlying system
96400 terminal support for this processing, are both conforming. Implementations are strongly urged
96401 to use the underlying system functionality, if at all possible, for compatibility with other system
96402 text input interfaces.

96403 Historically, when the *eof* character was used to decrement the **autoindent** level, the cursor
96404 moved to display the new end of the **autoindent** characters, but did not move the cursor to a
96405 new line, nor did it erase the <control>-D character from the line. POSIX.1-2024 does not specify
96406 that the cursor remain on the same line or that the rest of the line is erased; however,
96407 implementations are strongly encouraged to provide the best possible user interface; that is, the
96408 cursor should remain on the same line, and any <control>-D character on the line should be
96409 erased.

96410 POSIX.1-2024 does not require the historical 4 BSD input editing character ``reprint'',
96411 traditionally <control>-R, which redisplayed the current input from the user. For this reason,
96412 and because the functionality cannot be implemented after the line has been terminated by the
96413 user, POSIX.1-2024 makes no requirements about this functionality. Implementations are
96414 strongly urged to make this historical functionality available, if possible.

96415 Historically, <control>-Q did not perform a literal next function in *ex*, as it did in *vi*.
96416 POSIX.1-2024 requires conformance to historical practice to avoid breaking historical *ex* scripts
96417 and **.exrc** files.

96418 **eof**

96419 Whether the *eof* character immediately modifies the **autoindent** characters in the prompt is left
96420 unspecified so that implementations can conform in the presence of systems that do not support
96421 this functionality. Implementations are encouraged to modify the line and redisplay it
96422 immediately, if possible.

96423 The specification of the handling of the *eof* character differs from historical practice only in that
96424 *eof* characters are not discarded if they follow normal characters in the text input. Historically,
96425 they were always discarded.

96426 **Command Descriptions in ex**

96427 Historically, several commands (for example, **global**, **v**, **visual**, **s**, **write**, **wq**, **yank**, **!**, **<**, **>**, **&**, and
96428 **~**) were executable in empty files (that is, the default address(es) were 0), or permitted explicit
96429 addresses of 0 (for example, 0 was a valid address, or 0,0 was a valid range). Addresses of 0, or
96430 command execution in an empty file, make sense only for commands that add new text to the
96431 edit buffer or write commands (because users may wish to write empty files). POSIX.1-2024
96432 requires this behavior for such commands and disallows it otherwise, for consistency and
96433 simplicity of specification.

96434 A count to an *ex* command has been historically corrected to be no greater than the last line in a
96435 file; for example, in a five-line file, the command **1,6print** would fail, but the command
96436 **1print300** would succeed. POSIX.1-2024 requires conformance to historical practice.

96437 Historically, the use of flags in *ex* commands could be obscure. General historical practice was as
96438 described by POSIX.1-2024, but there were some special cases. For instance, the **list**, **number**,
96439 and **print** commands ignored trailing address offsets; for example, **3p +++#** would display line
96440 3, and 3 would be the current line after the execution of the command. The **open** and **visual**
96441 commands ignored both the trailing offsets and the trailing flags. Also, flags specified to the
96442 **open** and **visual** commands interacted badly with the **list** edit option, and setting and then
96443 unsetting it during the open/visual session would cause *vi* to stop displaying lines in the
96444 specified format. For consistency and simplicity of specification, POSIX.1-2024 does not permit
96445 any of these exceptions to the general rule.

96446 POSIX.1-2024 uses the word *copy* in several places when discussing buffers. This is not intended
96447 to imply implementation.

96448 Historically, *ex* users could not specify numeric buffers because of the ambiguity this would
96449 cause; for example, in the command **3 delete 2**, it is unclear whether 2 is a buffer name or a
96450 *count*. POSIX.1-2024 requires conformance to historical practice by default, but does not
96451 preclude extensions.

96452 Historically, the contents of the unnamed buffer were frequently discarded after commands that
96453 did not explicitly affect it; for example, when using the **edit** command to switch files. For
96454 consistency and simplicity of specification, POSIX.1-2024 does not permit this behavior.

96455 The *ex* utility did not historically have access to the numeric buffers, and, furthermore, deleting
96456 lines in *ex* did not modify their contents. For example, if, after doing a delete in *vi*, the user
96457 switched to *ex*, did another delete, and then switched back to *vi*, the contents of the numeric
96458 buffers would not have changed. POSIX.1-2024 requires conformance to historical practice.
96459 Numeric buffers are described in the *ex* utility in order to confine the description of buffers to a
96460 single location in POSIX.1-2024.

96461 The metacharacters that trigger shell expansion in *file* arguments match historical practice, as
96462 does the method for doing shell expansion. Implementations wishing to provide users with the
96463 flexibility to alter the set of metacharacters are encouraged to provide a **shellmeta** string edit

option.

Historically, *ex* commands executed from *vi* refreshed the screen when it did not strictly need to do so; for example, **:!date > /dev/null** does not require a screen refresh because the output of the UNIX *date* command requires only a single line of the screen. POSIX.1-2024 requires that the screen be refreshed if it has been overwritten, but makes no requirements as to how an implementation should make that determination. Implementations may prompt and refresh the screen regardless.

Abbreviate

Historical practice was that characters that were entered as part of an abbreviation replacement were subject to **map** expansions, the **showmatch** edit option, further abbreviation expansions, and so on; that is, they were logically pushed onto the terminal input queue, and were not a simple replacement. POSIX.1-2024 requires conformance to historical practice. Historical practice was that whenever a non-word character (that had not been escaped by a <control>-V) was entered after a word character, *vi* would check for abbreviations. The check was based on the type of the character entered before the word character of the word/non-word pair that triggered the check. The word character of the word/non-word pair that triggered the check and all characters entered before the trigger pair that were of that type were included in the check, with the exception of <blank> characters, which always delimited the abbreviation.

This means that, for the abbreviation to work, the *lhs* must end with a word character, there can be no transitions from word to non-word characters (or *vice versa*) other than between the last and next-to-last characters in the *lhs*, and there can be no <blank> characters in the *lhs*. In addition, because of the historical quoting rules, it was impossible to enter a literal <control>-V in the *lhs*. POSIX.1-2024 requires conformance to historical practice. Historical implementations did not inform users when abbreviations that could never be used were entered; implementations are strongly encouraged to do so.

For example, the following abbreviations will work:

```
:ab (p  REPLACE
:ab p   REPLACE
:ab ((p REPLACE
```

The following abbreviations will not work:

```
:ab (   REPLACE
:ab (pp REPLACE
```

Historical practice is that words on the *vi* colon command line were subject to abbreviation expansion, including the arguments to the **abbrev** (and more interestingly) the **unabbrev** command. Because there are implementations that do not do abbreviation expansion for the first argument to those commands, this is permitted, but not required, by POSIX.1-2024. However, the following sequence:

```
:ab foo bar
:ab foo baz
```

resulted in the addition of an abbreviation of `"baz"` for the string `"bar"` in historical *ex*/*vi*, and the sequence:

```
:ab foo1 bar
:ab foo2 bar
:unabbreviate foo2
```

deleted the abbreviation `"foo1"`, not `"foo2"`. These behaviors are not permitted by

96509 POSIX.1-2024 because they clearly violate the expectations of the user.

96510 It was historical practice that <control>-V, not <backslash>, characters be interpreted as escaping
96511 subsequent characters in the **abbreviate** command. POSIX.1-2024 requires conformance to
96512 historical practice; however, it should be noted that an abbreviation containing a <blank> will
96513 never work.

96514 **Append**

96515 Historically, any text following a <vertical-line> command separator after an **append**, **change**, or
96516 **insert** command became part of the insert text. For example, in the command:

96517 `:g/pattern/append|stuff1`

96518 a line containing the text `"stuff1"` would be appended to each line matching pattern. It was
96519 also historically valid to enter:

96520 `:append|stuff1`
96521 `stuff2`
96522 `.`

96523 and the text on the *ex* command line would be appended along with the text inserted after it.
96524 There was an historical bug, however, that the user had to enter two terminating lines (the `'.'`
96525 lines) to terminate text input mode in this case. POSIX.1-2024 requires conformance to historical
96526 practice, but disallows the historical need for multiple terminating lines.

96527 **Change**

96528 See the RATIONALE for the **append** command. Historical practice for cursor positioning after
96529 the change command when no text is input, is as described in POSIX.1-2024. However, one
96530 System V implementation is known to have been modified such that the cursor is positioned on
96531 the first address specified, and not on the line before the first address. POSIX.1-2024 disallows
96532 this modification for consistency.

96533 Historically, the **change** command did not support buffer arguments, although some
96534 implementations allow the specification of an optional buffer. This behavior is neither required
96535 nor disallowed by POSIX.1-2024.

96536 **Change Directory**

96537 A common extension in *ex* implementations is to use the elements of a **cdpath** edit option as
96538 prefix directories for *path* arguments to **chdir** that are relative pathnames and that do not have
96539 `'.'` or `".."` as their first component. Elements in the **cdpath** edit option are <colon>-separated.
96540 The initial value of the **cdpath** edit option is the value of the shell *CDPATH* environment
96541 variable. This feature was not included in POSIX.1-2024 because it does not exist in any of the
96542 implementations considered historical practice.

96543 **Copy**

96544 Historical implementations of *ex* permitted copies to lines inside of the specified range; for
96545 example, **:2,5copy3** was a valid command. POSIX.1-2024 requires conformance to historical
96546 practice.

96547 **Delete**

96548 POSIX.1-2024 requires support for the historical parsing of a **delete** command followed by flags,
96549 without any intervening <blank> characters. For example:

96550 **1dp** Deletes the first line and prints the line that was second.

96551 **1delep** As for **1dp**.

96552 **1d** Deletes the first line, saving it in buffer *p*.

96553 **1d p1l** (Pee-one-ell.) Deletes the first line, saving it in buffer *p*, and listing the line that was
96554 second.

96555 **Edit**

96556 Historically, any *ex* command could be entered as a **+***command* argument to the **edit** command,
96557 although some (for example, **insert** and **append**) were known to confuse historical
96558 implementations. For consistency and simplicity of specification, POSIX.1-2024 requires that any
96559 command be supported as an argument to the **edit** command.

96560 Historically, the command argument was executed with the current line set to the last line of the
96561 file, regardless of whether the **edit** command was executed from visual mode or not.
96562 POSIX.1-2024 requires conformance to historical practice.

96563 Historically, the **+***command* specified to the **edit** and **next** commands was delimited by the first
96564 <blank>, and there was no way to quote them. For consistency, POSIX.1-2024 requires that the
96565 usual *ex* backslash quoting be provided.

96566 Historically, specifying the **+***command* argument to the edit command required a filename to be
96567 specified as well; for example, **:edit +100** would always fail. For consistency and simplicity of
96568 specification, POSIX.1-2024 does not permit this usage to fail for that reason.

96569 Historically, only the cursor position of the last file edited was remembered by the editor.
96570 POSIX.1-2024 requires that this be supported; however, implementations are permitted to
96571 remember and restore the cursor position for any file previously edited.

96572 **File**

96573 Historical versions of the *ex* editor **file** command displayed a current line and number of lines in
96574 the edit buffer of 0 when the file was empty, while the *vi* <control>-G command displayed a
96575 current line and number of lines in the edit buffer of 1 in the same situation. POSIX.1-2024 does
96576 not permit this discrepancy, instead requiring that a message be displayed indicating that the file
96577 is empty.

96578 **Global**

96579 The two-pass operation of the **global** and **v** commands is not intended to imply implementation,
96580 only the required result of the operation.

96581 The current line and column are set as specified for the individual *ex* commands. This
96582 requirement is cumulative; that is, the current line and column must track across all the
96583 commands executed by the **global** or **v** commands.

96584 **Insert**

96585 See the RATIONALE for the **append** command.

96586 Historically, **insert** could not be used with an address of zero; that is, not when the edit buffer
96587 was empty. POSIX.1-2024 requires that this command behave consistently with the **append**
96588 command.

96589 **Join**

96590 The action of the **join** command in relation to the special characters is only defined for the
96591 POSIX locale because the correct amount of white space after a period varies; in Japanese none is
96592 required, in French only a single space, and so on.

96593 **List**

96594 The historical output of the **list** command was potentially ambiguous. The standard developers
96595 believed correcting this to be more important than adhering to historical practice, and
96596 POSIX.1-2024 requires unambiguous output.

96597 **Map**

96598 Historically, command mode maps only applied to command names; for example, if the
96599 character `'x'` was mapped to `'y'`, the command **fx** searched for the `'x'` character, not the `'y'`
96600 character. POSIX.1-2024 requires this behavior. Historically, entering <control>-V as the first
96601 character of a *vi* command was an error. Several implementations have extended the semantics
96602 of *vi* such that <control>-V means that the subsequent command character is not mapped. This
96603 is permitted, but not required, by POSIX.1-2024. Regardless, using <control>-V to escape the
96604 second or later character in a sequence of characters that might match a **map** command, or any
96605 character in text input mode, is historical practice, and stops the entered keys from matching a
96606 map. POSIX.1-2024 requires conformance to historical practice.

96607 Historical implementations permitted digits to be used as a **map** command *lhs*, but then ignored
96608 the map. POSIX.1-2024 requires that the mapped digits not be ignored.

96609 The historical implementation of the **map** command did not permit **map** commands that were
96610 more than a single character in length if the first character was printable. This behavior is
96611 permitted, but not required, by POSIX.1-2024.

96612 Historically, mapped characters were remapped unless the **remap** edit option was not set, or the
96613 prefix of the mapped characters matched the mapping characters; for example, in the **map**:

96614 ```
:map ab abcd
```

96615 the characters `"ab"` were used as is and were not remapped, but the characters `"cd"` were
96616 mapped if appropriate. This can cause infinite loops in the *vi* mapping mechanisms.
96617 POSIX.1-2024 requires conformance to historical practice, and that such loops be interruptible.

96618 Text input maps had the same problems with expanding the *lhs* for the *ex* **map!** and **unmap!**
96619 command as did the *ex* **abbreviate** and **unabbreviate** commands. See the RATIONALE for the *ex*
96620 **abbreviate** command. POSIX.1-2024 requires similar modification of some historical practice for
96621 the **map** and **unmap** commands, as described for the **abbreviate** and **unabbreviate** commands.

96622 Historically, **map**s that were subsets of other **map**s behaved differently depending on the order
96623 in which they were defined. For example:

96624 ```
:map! ab     short
96625 :map! abc    long
```


would always translate the characters `"ab"` to `"short"`, regardless of how fast the characters `"abc"` were entered. If the entry order was reversed:

```
:map! abc    long
:map! ab     short
```

the characters `"ab"` would cause the editor to pause, waiting for the completing `'c'` character, and the characters might never be mapped to `"short"`. For consistency and simplicity of specification, POSIX.1-2024 requires that the shortest match be used at all times.

The length of time the editor spends waiting for the characters to complete the *lhs* is unspecified because the timing capabilities of systems are often inexact and variable, and it may depend on other factors such as the speed of the connection. The time should be long enough for the user to be able to complete the sequence, but not long enough for the user to have to wait. Some implementations of *vi* have added a **keytime** option, which permits users to set the number of 0,1 seconds the editor waits for the completing characters. Because mapped terminal function and cursor keys tend to start with an <ESC> character, and <ESC> is the key ending *vi* text input mode, **map**s starting with <ESC> characters are generally exempted from this timeout period, or, at least timed out differently.

Mark

Historically, users were able to set the ``previous context'' marks explicitly. In addition, the *ex* commands **'** and **`** and the *vi* commands **'**, **`**, **`'**, and **'`** all referred to the same mark. In addition, the previous context marks were not set if the command, with which the address setting the mark was associated, failed. POSIX.1-2024 requires conformance to historical practice. Historically, if marked lines were deleted, the mark was also deleted, but would reappear if the change was undone. POSIX.1-2024 requires conformance to historical practice.

The description of the special events that set the **`** and **'** marks matches historical practice. For example, historically the command **/a/,/b/** did not set the **`** and **'** marks, but the command **/a/,/b/delete** did.

Next

Historically, any *ex* command could be entered as a **+***command* argument to the **next** command, although some (for example, **insert** and **append**) were known to confuse historical implementations. POSIX.1-2024 requires that any command be permitted and that it behave as specified. The **next** command can accept more than one file, so usage such as:

```
next `ls [abc] `
```

is valid; it need not be valid for the **edit** or **read** commands, for example, because they expect only one filename.

Historically, the **next** command behaved differently from the **:rewind** command in that it ignored the force flag if the **autowrite** flag was set. For consistency, POSIX.1-2024 does not permit this behavior.

Historically, the **next** command positioned the cursor as if the file had never been edited before, regardless. POSIX.1-2024 does not permit this behavior, for consistency with the **edit** command.

Implementations wanting to provide a counterpart to the **next** command that edited the previous file have used the command **prev[ious],** which takes no *file* argument. POSIX.1-2024 does not require this command.

96668 **Open**

96669 Historically, the **open** command would fail if the **open** edit option was not set. POSIX.1-2024
96670 does not mention the **open** edit option and does not require this behavior. Some historical
96671 implementations do not permit entering open mode from open or visual mode, only from *ex*
96672 mode. For consistency, POSIX.1-2024 does not permit this behavior.

96673 Historically, entering open mode from the command line (that is, *vi* **+open**) resulted in
96674 anomalous behaviors; for example, the *ex* file and *set* commands, and the *vi* command
96675 <control>-G did not work. For consistency, POSIX.1-2024 does not permit this behavior.

96676 Historically, the **open** command only permitted `'/'` characters to be used as the search pattern
96677 delimiter. For consistency, POSIX.1-2024 requires that the search delimiters used by the **s**, **global**,
96678 and **v** commands be accepted as well.

96679 **Preserve**

96680 The **preserve** command does not historically cause the file to be considered unmodified for the
96681 purposes of future commands that may exit the editor. POSIX.1-2024 requires conformance to
96682 historical practice.

96683 Historical documentation stated that mail was not sent to the user when preserve was executed;
96684 however, historical implementations did send mail in this case. POSIX.1-2024 requires
96685 conformance to the historical implementations.

96686 **Print**

96687 The writing of NUL by the **print** command is not specified as a special case because the standard
96688 developers did not want to require *ex* to support NUL characters. Historically, characters were
96689 displayed using the ARPA standard mappings, which are as follows:

96690 1. Printable characters are left alone.

96691 2. Control characters less than \177 are represented as `'^'` followed by the character offset
96692 from the `'@'` character in the ASCII map; for example, \007 is represented as `'^G'`.

96693 3. \177 is represented as `'^'` followed by `'?'`.

96694 The display of characters having their eighth bit set was less standard. Existing implementations
96695 use hex (0x00), octal (\000), and a meta-bit display. (The latter displayed bytes that had their
96696 eighth bit set as the two characters `"M-"` followed by the seven-bit display as described above.)
96697 The latter probably has the best claim to historical practice because it was used for the **−v** option
96698 of 4 BSD and 4 BSD-derived versions of the *cat* utility since 1980.

96699 No specific display format is required by POSIX.1-2024.

96700 Explicit dependence on the ASCII character set has been avoided where possible, hence the use
96701 of the phrase an ``implementation-defined multi-character sequence'' for the display of non-
96702 printable characters in preference to the historical usage of, for instance, `"^I"` for the <tab>.
96703 Implementations are encouraged to conform to historical practice in the absence of any strong
96704 reason to diverge.

96705 Historically, all *ex* commands beginning with the letter `'p'` could be entered using capitalized
96706 versions of the commands; for example, **P[rint]**, **Pre[serve]**, and **Pu[t]** were all valid command
96707 names. POSIX.1-2024 permits, but does not require, this historical practice because capital forms
96708 of the commands are used by some implementations for other purposes.

96709 **Put**

96710 Historically, an *ex* **put** command, executed from open or visual mode, was the same as the open
96711 or visual mode **P** command, if the buffer was named and was cut in character mode, and the
96712 same as the **p** command if the buffer was named and cut in line mode. If the unnamed buffer
96713 was the source of the text, the entire line from which the text was taken was usually **put**, and the
96714 buffer was handled as if in line mode, but it was possible to get extremely anomalous behavior.
96715 In addition, using the **Q** command to switch into *ex* mode, and then doing a **put** often resulted in
96716 errors as well, such as appending text that was unrelated to the (supposed) contents of the
96717 buffer. For consistency and simplicity of specification, POSIX.1-2024 does not permit these
96718 behaviors. All *ex* **put** commands are required to operate in line mode, and the contents of the
96719 buffers are not altered by changing the mode of the editor.

96720 **Read**

96721 Historically, an *ex* **read** command executed from open or visual mode, executed in an empty file,
96722 left an empty line as the first line of the file. For consistency and simplicity of specification,
96723 POSIX.1-2024 does not permit this behavior. Historically, a **read** in open or visual mode from a
96724 program left the cursor at the last line read in, not the first. For consistency, POSIX.1-2024 does
96725 not permit this behavior.

96726 Historical implementations of *ex* were unable to undo **read** commands that read from the output
96727 of a program. For consistency, POSIX.1-2024 does not permit this behavior.

96728 Historically, the *ex* and *vi* message after a successful **read** or **write** command specified
96729 ``characters'', not ``bytes''. POSIX.1-2024 requires that the number of bytes be displayed, not the
96730 number of characters, because it may be difficult in multi-byte implementations to determine the
96731 number of characters read. Implementations are encouraged to clarify the message displayed to
96732 the user.

96733 Historically, reads were not permitted on files other than type regular, except that FIFO files
96734 could be read (probably only because they did not exist when *ex* and *vi* were originally written).
96735 Because the historical *ex* evaluated **read!** and **read !** equivalently, there can be no optional way
96736 to force the read. POSIX.1-2024 permits, but does not require, this behavior.

96737 **Recover**

96738 Some historical implementations of the editor permitted users to recover the edit buffer contents
96739 from a previous edit session, and then exit without saving those contents (or explicitly
96740 discarding them). The intent of POSIX.1-2024 in requiring that the edit buffer be treated as
96741 already modified is to prevent this user error.

96742 **Rewind**

96743 Historical implementations supported the **rewind** command when the user was editing the first
96744 file in the list; that is, the file that the **rewind** command would edit. POSIX.1-2024 requires
96745 conformance to historical practice.

96746 **Substitute**

96747 Historically, *ex* accepted an **r** option to the **s** command. The effect of the **r** option was to use the
96748 last regular expression used in any command as the pattern, the same as the ~ command. The **r**
96749 option is not required by POSIX.1-2024. Historically, the **c** and **g** options were toggled; for
96750 example, the command **:s/abc/def/** was the same as **s/abc/def/ccccgggg**. For simplicity of
96751 specification, POSIX.1-2024 does not permit this behavior.

96752 The tilde command is often used to replace the last search RE. For example, in the sequence:

96753 `s/red/blue/`
96754 `/green`
96755 `~`

96756 the ~ command is equivalent to:

96757 `s/green/blue/`

96758 Historically, *ex* accepted all of the following forms:

96759 `s/abc/def/`
96760 `s/abc/def`
96761 `s/abc/`
96762 `s/abc`

96763 POSIX.1-2024 requires conformance to this historical practice.

96764 The **s** command presumes that the `'^'` character only occupies a single column in the display.
96765 Much of the *ex* and *vi* specification presumes that the <space> only occupies a single column in
96766 the display. There are no known character sets for which this is not true.

96767 Historically, the final column position for the substitute commands was based on previous
96768 column movements; a search for a pattern followed by a substitution would leave the column
96769 position unchanged, while a 0 command followed by a substitution would change the column
96770 position to the first non-<blank>. For consistency and simplicity of specification, POSIX.1-2024
96771 requires that the final column position always be set to the first non-<blank>.

96772 **Set**

96773 Historical implementations redisplayed all of the options for each occurrence of the **all** keyword.
96774 POSIX.1-2024 permits, but does not require, this behavior.

96775 **Tag**

96776 No requirement is made as to where *ex* and *vi* shall look for the file referenced by the tag entry.
96777 Historical practice has been to look for the path found in the **tags** file, based on the current
96778 directory. A useful extension found in some implementations is to look based on the directory
96779 containing the tags file that held the entry, as well. No requirement is made as to which reference
96780 for the tag in the tags file is used. This is deliberate, in order to permit extensions such as
96781 multiple entries in a tags file for a tag.

96782 Because users often specify many different tags files, some of which need not be relevant or exist
96783 at any particular time, POSIX.1-2024 requires that error messages about problem tags files be
96784 displayed only if the requested tag is not found, and then, only once for each time that the **tag**
96785 edit option is changed.

96786 The requirement that the current edit buffer be unmodified is only necessary if the file indicated
96787 by the tag entry is not the same as the current file (as defined by the current pathname).
96788 Historically, the file would be reloaded if the filename had changed, as well as if the filename

was different from the current pathname. For consistency and simplicity of specification, POSIX.1-2024 does not permit this behavior, requiring that the name be the only factor in the decision.

Historically, *vi* only searched for tags in the current file from the current cursor to the end of the file, and therefore, if the **wrapscan** option was not set, tags occurring before the current cursor were not found. POSIX.1-2024 considers this a bug, and implementations are required to search for the first occurrence in the file, regardless.

Undo

The **undo** description deliberately uses the word ``modified''. The **undo** command is not intended to undo commands that replace the contents of the edit buffer, such as **edit**, **next**, **tag**, or **recover**.

Cursor positioning after the **undo** command was inconsistent in the historical *vi*, sometimes attempting to restore the original cursor position (**global**, **undo**, and **v** commands), and sometimes, in the presence of maps, placing the cursor on the last line added or changed instead of the first. POSIX.1-2024 requires a simplified behavior for consistency and simplicity of specification.

Version

The **version** command cannot be exactly specified since there is no widely-accepted definition of what the version information should contain. Implementations are encouraged to do something reasonably intelligent.

Write

Historically, the *ex* and *vi* message after a successful **read** or **write** command specified ``characters'', not ``bytes''. POSIX.1-2024 requires that the number of bytes be displayed, not the number of characters because it may be difficult in multi-byte implementations to determine the number of characters written. Implementations are encouraged to clarify the message displayed to the user.

Implementation-defined tests are permitted so that implementations can make additional checks; for example, for locks or file modification times.

Historically, attempting to append to a nonexistent file caused an error. It has been left unspecified in POSIX.1-2024 to permit implementations to let the **write** succeed, so that the append semantics are similar to those of the historical *csh*.

Historical *vi* permitted empty edit buffers to be written. However, since the way *vi* got around dealing with ``empty'' files was to always have a line in the edit buffer, no matter what, it wrote them as files of a single, empty line. POSIX.1-2024 does not permit this behavior.

Historically, *ex* restored standard output and standard error to their values as of when *ex* was invoked, before writes to programs were performed. This could disturb the terminal configuration as well as be a security issue for some terminals. POSIX.1-2024 does not permit this, requiring that the program output be captured and displayed as if by the *ex* **print** command.

Adjust Window

Historically, the line count was set to the value of the **scroll** option if the type character was end-of-file. This feature was broken on most historical implementations long ago, however, and is not documented anywhere. For this reason, POSIX.1-2024 is resolutely silent.

Historically, the **z** command was <blank>-sensitive and **z +** and **z** – did different things than **z+** and **z**– because the type could not be distinguished from a flag. (The commands **z .** and **z =** were historically invalid.) POSIX.1-2024 requires conformance to this historical practice.

Historically, the **z** command was further <blank>-sensitive in that the *count* could not be <blank>-delimited; for example, the commands **z= 5** and **z– 5** were also invalid. Because the *count* is not ambiguous with respect to either the type character or the flags, this is not permitted by POSIX.1-2024.

Escape

Historically, *ex* filter commands only read the standard output of the commands, letting standard error appear on the terminal as usual. The *vi* utility, however, read both standard output and standard error. POSIX.1-2024 requires the latter behavior for both *ex* and *vi*, for consistency.

In Issue 8 the *system*() function was changed to require that the POSIX shell be invoked with `"sh"`, `"-c"`, `"--"`, and *command* arguments to make it easier to execute programs with <hyphen-minus> (`'-'`) or <plus-sign> (`'+'`) as the first character of the program's filename. A similar request to have the *ex* **escape** command do the same was not accepted. Unlike *system*() (which always invokes a POSIX shell), *ex* invokes the program named by the **shell** edit option. For example, the *csh* and *tcsh* shells that are frequently used as login shells do not recognize `"--"` after `"-c"` as an end-of-options indicator. The program need not even be one that recognizes any POSIX shell command line syntax. Some users invoke shell scripts to process lines that are being supplied to the specified utility. These utilities know that they will be given `"-c"` as a first argument and just ignore it. Any utilities used in this manner would have to be modified to skip over another argument (the `"--"`) to find the desired argument.

Shift Left and Shift Right

Historically, it was possible to add shift characters to increase the effect of the command; for example, **<<<** outdented (or **>>>** indented) the lines 3 levels of indentation instead of the default 1. POSIX.1-2024 requires conformance to historical practice.

<control>-D

Historically, the <control>-D command erased the prompt, providing the user with an unbroken presentation of lines from the edit buffer. This is not required by POSIX.1-2024; implementations are encouraged to provide it if possible. Historically, the <control>-D command took, and then ignored, a *count*. POSIX.1-2024 does not permit this behavior.

96864 **Write Line Number**

96865 Historically, the *ex* **=** command, when executed in *ex* mode in an empty edit buffer, reported 0,
96866 and from open or visual mode, reported 1. For consistency and simplicity of specification,
96867 POSIX.1-2024 does not permit this behavior.

96868 **Execute**

96869 Historically, *ex* did not correctly handle the inclusion of text input commands (that is, **append**,
96870 **insert**, and **change**) in executed buffers. POSIX.1-2024 does not permit this exclusion for
96871 consistency.

96872 Historically, the logical contents of the buffer being executed did not change if the buffer itself
96873 were modified by the commands being executed; that is, buffer execution did not support self-
96874 modifying code. POSIX.1-2024 requires conformance to historical practice.

96875 Historically, the @ command took a range of lines, and the @ buffer was executed once per line,
96876 with the current line (`'.'`) set to each specified line. POSIX.1-2024 requires conformance to
96877 historical practice.

96878 Some historical implementations did not notice if errors occurred during buffer execution. This,
96879 coupled with the ability to specify a range of lines for the *ex* @ command, makes it trivial to
96880 cause them to drop **core**. POSIX.1-2024 requires that implementations stop buffer execution if
96881 any error occurs, if the specified line doesn't exist, or if the contents of the edit buffer itself are
96882 replaced (for example, the buffer executes the *ex* **:edit** command).

96883 **Regular Expressions in ex**

96884 Historical practice is that the characters in the replacement part of the last **s** command—that is,
96885 those matched by entering a `'~'` in the regular expression—were not further expanded by the
96886 regular expression engine. So, if the characters contained the string `"a.,"` they would match
96887 `'a'` followed by `"."` and not `'a'` followed by any character. POSIX.1-2024 requires
96888 conformance to historical practice.

96889 **Edit Options in ex**

96890 The following paragraphs describe the historical behavior of some edit options that were not, for
96891 whatever reason, included in POSIX.1-2024. Implementations are strongly encouraged to only
96892 use these names if the functionality described here is fully supported.

96893 **extended** The **extended** edit option has been used in some implementations of *vi* to provide
96894 extended regular expressions instead of basic regular expressions This option was
96895 omitted from POSIX.1-2024 because it is not widespread historical practice.

96896 **flash** The **flash** edit option historically caused the screen to flash instead of beeping on
96897 error. This option was omitted from POSIX.1-2024 because it is not found in some
96898 historical implementations.

96899 **hardtabs** The **hardtabs** edit option historically defined the number of columns between
96900 hardware tab settings. This option was omitted from POSIX.1-2024 because it was
96901 believed to no longer be generally useful.

96902 **modeline** The **modeline** (sometimes named **modelines**) edit option historically caused *ex* or
96903 *vi* to read the five first and last lines of the file for editor commands. This option is
96904 a security problem, and vendors are strongly encouraged to delete it from
96905 historical implementations.

open The **open** edit option historically disallowed the *ex* **open** and **visual** commands. This edit option was omitted because these commands are required by POSIX.1-2024.

optimize The **optimize** edit option historically expedited text throughput by setting the terminal to not do automatic <carriage-return> characters when printing more than one logical line of output. This option was omitted from POSIX.1-2024 because it was intended for terminals without addressable cursors, which are rarely, if ever, still used.

ruler The **ruler** edit option has been used in some implementations of *vi* to present a current row/column ruler for the user. This option was omitted from POSIX.1-2024 because it is not widespread historical practice.

sourceany The **sourceany** edit option historically caused *ex* or *vi* to source start-up files that were owned by users other than the user running the editor. This option is a security problem, and vendors are strongly encouraged to remove it from their implementations.

timeout The **timeout** edit option historically enabled the (now standard) feature of only waiting for a short period before returning keys that could be part of a macro. This feature was omitted from POSIX.1-2024 because its behavior is now standard, it is not widely useful, and it was rarely documented.

verbose The **verbose** edit option has been used in some implementations of *vi* to cause *vi* to output error messages for common errors; for example, attempting to move the cursor past the beginning or end of the line instead of only alerting the screen. (The historical *vi* only alerted the terminal and presented no message for such errors. The historical editor option **terse** did not select when to present error messages, it only made existing error messages more or less verbose.) This option was omitted from POSIX.1-2024 because it is not widespread historical practice; however, implementors are encouraged to use it if they wish to provide error messages for naive users.

wraplen The **wraplen** edit option has been used in some implementations of *vi* to specify an automatic margin measured from the left margin instead of from the right margin. This is useful when multiple screen sizes are being used to edit a single file. This option was omitted from POSIX.1-2024 because it is not widespread historical practice; however, implementors are encouraged to use it if they add this functionality.

autoindent, ai

Historically, the command **0a** did not do any autoindentation, regardless of the current indentation of line 1. POSIX.1-2024 requires that any indentation present in line 1 be used.

autoprint, ap

Historically, the **autoprint** edit option was not completely consistent or based solely on modifications to the edit buffer. Exceptions were the **read** command (when reading from a file, but not from a filter), the **append**, **change**, **insert**, **global**, and **v** commands, all of which were not affected by **autoprint**, and the **tag** command, which was affected by **autoprint**. POSIX.1-2024 requires conformance to historical practice.

Historically, the **autoprint** option only applied to the last of multiple commands entered using <vertical-line> delimiters; for example, **delete** <newline> was affected by **autoprint**, but

96951 **delete|version** <newline> was not. POSIX.1-2024 requires conformance to historical practice.

96952 **autowrite, aw**

96953 Appending the `'!'` character to the *ex* **next** command to avoid performing an automatic write
96954 was not supported in historical implementations. POSIX.1-2024 requires that the behavior match
96955 the other *ex* commands for consistency.

96956 **ignorecase, ic**

96957 Historical implementations of case-insensitive matching (the **ignorecase** edit option) lead to
96958 counter-intuitive situations when uppercase characters were used in range expressions.
96959 Historically, the process was as follows:

96960 1. Take a line of text from the edit buffer.
96961 2. Convert uppercase to lowercase in text line.
96962 3. Convert uppercase to lowercase in regular expressions, except in character class
96963 specifications.
96964 4. Match regular expressions against text.

96965 This would mean that, with **ignorecase** in effect, the text:

96966 `The cat sat on the mat`

96967 would be matched by

96968 `/^the/`

96969 but not by:

96970 `/^[A-Z]he/`

96971 For consistency with other commands implementing regular expressions, POSIX.1-2024 does not
96972 permit this behavior.

96973 **paragraphs, para**

96974 The ISO POSIX-2:1993 standard made the default **paragraphs** and **sections** edit options
96975 implementation-defined, arguing they were historically oriented to the UNIX system *troff* text
96976 formatter, and a ``portable user'' could use the {, }, [[,]], **(**, and **)** commands in open or visual
96977 mode and have the cursor stop in unexpected places. POSIX.1-2024 specifies their values in the
96978 POSIX locale because the unusual grouping (they only work when grouped into two characters
96979 at a time) means that they cannot be used for general-purpose movement, regardless.

96980 **readonly**

96981 Implementations are encouraged to provide the best possible information to the user as to the
96982 read-only status of the file, with the exception that they should not consider the current special
96983 privileges of the process. This provides users with a safety net because they must force the
96984 overwrite of read-only files, even when running with additional privileges.

96985 The **readonly** edit option specification largely conforms to historical practice. The only
96986 difference is that historical implementations did not notice that the user had set the **readonly**
96987 edit option in cases where the file was already marked read-only for some reason, and would
96988 therefore reinitialize the **readonly** edit option the next time the contents of the edit buffer were
96989 replaced. This behavior is disallowed by POSIX.1-2024.

report

The requirement that lines copied to a buffer interact differently than deleted lines is historical practice. For example, if the **report** edit option is set to 3, deleting 3 lines will cause a report to be written, but 4 lines must be copied before a report is written.

The requirement that the *ex* **global**, **v**, **open**, **undo**, and **visual** commands present reports based on the total number of lines added or deleted during the command execution, and that commands executed by the **global** and **v** commands not present reports, is historical practice. POSIX.1-2024 extends historical practice by requiring that buffer execution be treated similarly. The reasons for this are two-fold. Historically, only the report by the last command executed from the buffer would be seen by the user, as each new report would overwrite the last. In addition, the standard developers believed that buffer execution had more in common with **global** and **v** commands than it did with other *ex* commands, and should behave similarly, for consistency and simplicity of specification.

showmatch, sm

The length of time the cursor spends on the matching character is unspecified because the timing capabilities of systems are often inexact and variable. The time should be long enough for the user to notice, but not long enough for the user to become annoyed. Some implementations of *vi* have added a **matchtime** option that permits users to set the number of 0,1 second intervals the cursor pauses on the matching character.

showmode

The **showmode** option has been used in some historical implementations of *ex* and *vi* to display the current editing mode when in open or visual mode. The editing modes have generally included ``command'' and ``input'', and sometimes other modes such as ``replace'' and ``change''. The string was usually displayed on the bottom line of the screen at the far right-hand corner. In addition, a preceding `'*'` character often denoted whether the contents of the edit buffer had been modified. The latter display has sometimes been part of the **showmode** option, and sometimes based on another option. This option was not available in the 4 BSD historical implementation of *vi*, but was viewed as generally useful, particularly to novice users, and is required by POSIX.1-2024.

The **smd** shorthand for the **showmode** option was not present in all historical implementations of the editor. POSIX.1-2024 requires it, for consistency.

Not all historical implementations of the editor displayed a mode string for command mode, differentiating command mode from text input mode by the absence of a mode string. POSIX.1-2024 permits this behavior for consistency with historical practice, but implementations are encouraged to provide a display string for both modes.

slowopen

Historically, the **slowopen** option was automatically set if the terminal baud rate was less than 1 200 baud, or if the baud rate was 1 200 baud and the **redraw** option was not set. The **slowopen** option had two effects. First, when inserting characters in the middle of a line, characters after the cursor would not be pushed ahead, but would appear to be overwritten. Second, when creating a new line of text, lines after the current line would not be scrolled down, but would appear to be overwritten. In both cases, ending text input mode would cause the screen to be refreshed to match the actual contents of the edit buffer. Finally, terminals that were sufficiently intelligent caused the editor to ignore the **slowopen** option. POSIX.1-2024 permits most historical behavior, extending historical practice to require **slowopen** behaviors if the edit option

is set by the user.

tags

The default path for tags files is left unspecified as implementations may have their own **tags** implementations that do not correspond to the historical ones. The default **tags** option value should probably at least include the file **./tags**.

term

Historical implementations of *ex* and *vi* ignored changes to the **term** edit option after the initial terminal information was loaded. This is permitted by POSIX.1-2024; however, implementations are encouraged to permit the user to modify their terminal type at any time.

terse

Historically, the **terse** edit option optionally provided a shorter, less descriptive error message, for some error messages. This is permitted, but not required, by POSIX.1-2024. Historically, most common visual mode errors (for example, trying to move the cursor past the end of a line) did not result in an error message, but simply alerted the terminal. Implementations wishing to provide messages for novice users are urged to do so based on the **edit** option **verbose**, and not **terse**.

window

In historical implementations, the default for the **window** edit option was based on the baud rate as follows:

1. If the baud rate was less than 1 200, the **edit** option **w300** set the window value; for example, the line:

   ```
   set w300=12
   ```

   would set the window option to 12 if the baud rate was less than 1 200.

2. If the baud rate was equal to 1 200, the **edit** option **w1200** set the window value.

3. If the baud rate was greater than 1 200, the **edit** option **w9600** set the window value.

The **w300**, **w1200**, and **w9600** options do not appear in POSIX.1-2024 because of their dependence on specific baud rates.

In historical implementations, the size of the window displayed by various commands was related to, but not necessarily the same as, the **window** edit option. For example, the size of the window was set by the *ex* command **visual 10**, but it did not change the value of the **window** edit option. However, changing the value of the **window** edit option did change the number of lines that were displayed when the screen was repainted. POSIX.1-2024 does not permit this behavior in the interests of consistency and simplicity of specification, and requires that all commands that change the number of lines that are displayed do it by setting the value of the **window** edit option.

97070 **wrapmargin, wm**

97071 Historically, the **wrapmargin** option did not affect maps inserting characters that also had
97072 associated *count*s; for example **:map K 5aABC DEF**. Unfortunately, there are widely used
97073 maps that depend on this behavior. For consistency and simplicity of specification,
97074 POSIX.1-2024 does not permit this behavior.

97075 Historically, **wrapmargin** was calculated using the column display width of all characters on the
97076 screen. For example, an implementation using `"^I"` to represent <tab> characters when the **list**
97077 edit option was set, where `'^'` and `'I'` each took up a single column on the screen, would
97078 calculate the **wrapmargin** based on a value of 2 for each <tab>. The **number** edit option
97079 similarly changed the effective length of the line as well. POSIX.1-2024 requires conformance to
97080 historical practice.

97081 Earlier versions of this standard allowed for implementations with bytes other than eight bits,
97082 but this has been modified in this version.

97083 **FUTURE DIRECTIONS**

97084 If this utility is directed to create a new directory entry that contains any bytes that have the
97085 encoded value of a <newline> character, implementations are encouraged to treat this as an
97086 error. A future version of this standard may require implementations to treat this as an error.

97087 **SEE ALSO**

97088 Section 2.9.1.4 (on page 2502), *ctags*, *ed*, *sed*, *sh*, *stty*, *vi*

97089 XBD Table 5-1 (on page 113), Chapter 8 (on page 167), Section 9.3 (on page 181), Section 12.2 (on
97090 page 215)

97091 XSH *access*( )

97092 **CHANGE HISTORY**

97093 First released in Issue 2.

97094 **Issue 5**

97095 The FUTURE DIRECTIONS section is added.

97096 **Issue 6**

97097 This utility is marked as part of the User Portability Utilities option.

97098 The obsolescent SYNOPSIS is removed, removing the **+***command* and – options.

97099 The following new requirements on POSIX implementations derive from alignment with the
97100 Single UNIX Specification:

97101 • In the **map** command description, the sequence **#***digit* is added.

97102 • The **directory**, **edcompatible**, **redraw**, and **slowopen** edit options are added.

97103 The *ex* utility is extensively changed for alignment with the IEEE P1003.2b draft standard. This
97104 includes changes as a result of the IEEE PASC Interpretations 1003.2 #31, #38, #49, #50, #51, #52,
97105 #55, #56, #57, #61, #62, #63, #64, #65, and #78.

97106 The **–l** option is removed.

97107 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/23 is applied, correcting a URL.

97108 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/8 is applied, making an editorial
97109 correction in the EXTENDED DESCRIPTION.

97110 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/9 is applied, removing text describing
97111 behavior on systems with bytes consisting of more than eight bits.

Issue 7

Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if an operand is '−'.

Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.

Austin Group Interpretation 1003.1-2001 #121 is applied, clarifying the *ex* **write** command.

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0093 [584] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1185 is applied, changing the SIGCONT entry in ASYNCHRONOUS EVENTS and adding a SIGWINCH entry.

Austin Group Defect 1251 is applied, clarifying the **set** command, changing ``addr'' to ``2addr'' in the **!** command synopsis, and adding spaces in some synopsis lines.

Austin Group Defect 1281 is applied, changing the description of the **substitute** command to clarify that it is an error if the substitution fails on every addressed line.

Austin Group Defect 1298 is applied, changing the CONSEQUENCES OF ERRORS section.

Austin Group Defect 1378 is applied, changing the description of the *LC_MESSAGES* environment variable.

Austin Group Defect 1529 is applied, changing the synopsis of the **escape** command and adding related paragraphs to the APPLICATION USAGE and RATIONALE sections.

Austin Group Defect 1642 is applied, changing the description of the **redraw** edit option.

Austin Group Defect 1662 is applied, clarifying requirements relating to delimiters in addresses and in **s** commands.

NAME

expand — convert tabs to spaces

SYNOPSIS

```
expand [-t tablist] [file...]
```

DESCRIPTION

The *expand* utility shall write files or the standard input to the standard output with <tab> characters replaced with one or more <space> characters needed to pad to the next tab stop. Any <backspace> characters shall be copied to the output and cause the column position count for tab stop calculations to be decremented; the column position count shall not be decremented below zero.

OPTIONS

The *expand* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

–t *tablist* Specify the tab stops. The application shall ensure that the argument *tablist* consists of either a single positive decimal integer or a list of tabstops. If a single number is given, tabs shall be set that number of column positions apart instead of the default 8.

If a list of tabstops is given, the application shall ensure that it consists of a list of two or more positive decimal integers, separated by <blank> or <comma> characters, in ascending order. The <tab> characters shall be set at those specific column positions. Each tab stop *N* shall be an integer value greater than zero, and the list is in strictly ascending order. This is taken to mean that, from the start of a line of output, tabbing to position *N* shall cause the next character output to be in the (*N*+1)th column position on that line.

In the event of *expand* having to process a <tab> at a position beyond the last of those specified in a multiple tab-stop list, the <tab> shall be replaced by a single <space> in the output.

OPERANDS

The following operand shall be supported:

file The pathname of a text file to be used as input.

STDIN

See the INPUT FILES section.

INPUT FILES

Input files shall be text files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *expand*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the processing of <tab> and <space> characters, and for the determination of the width in column positions each character would

97183 occupy on an output device.

97184 *LC_MESSAGES*
97185 Determine the locale that should be used to affect the format and contents of
97186 diagnostic messages written to standard error.

97187 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

97188 **ASYNCHRONOUS EVENTS**
97189 Default.

97190 **STDOUT**
97191 The standard output shall be equivalent to the input files with <tab> characters converted into
97192 the appropriate number of <space> characters.

97193 **STDERR**
97194 The standard error shall be used only for diagnostic messages.

97195 **OUTPUT FILES**
97196 None.

97197 **EXTENDED DESCRIPTION**
97198 None.

97199 **EXIT STATUS**
97200 The following exit values shall be returned:

97201 0 Successful completion

97202 >0 An error occurred.

97203 **CONSEQUENCES OF ERRORS**
97204 The *expand* utility shall terminate with an error message and non-zero exit status upon
97205 encountering difficulties accessing one of the *file* operands.

97206 **APPLICATION USAGE**
97207 None.

97208 **EXAMPLES**
97209 None.

97210 **RATIONALE**
97211 The *expand* utility is useful for preprocessing text files (before sorting, looking at specific
97212 columns, and so on) that contain <tab> characters.

97213 See XBD Section 3.75 (on page 42).

97214 The *tablist* option-argument consists of integers in ascending order. Utility Syntax Guideline 8
97215 mandates that *expand* shall accept the integers (within the single argument) separated using
97216 either <comma> or <blank> characters.

97217 Earlier versions of this standard allowed the following form in the SYNOPSIS:

97218 `expand [-tabstop][-tab1,tab2,...,tabn][file ...]`

97219 This form is no longer specified by POSIX.1-2024 but may be present in some implementations.

97220 **FUTURE DIRECTIONS**
97221 None.

97222 **SEE ALSO**

97223 *tabs*, *unexpand*

97224 XBD Section 3.75 (on page 42), Chapter 8 (on page 167), Section 12.2 (on page 215)

97225 **CHANGE HISTORY**

97226 First released in Issue 4.

97227 **Issue 6**

97228 This utility is marked as part of the User Portability Utilities option.

97229 The APPLICATION USAGE section is added.

97230 The obsolescent SYNOPSIS is removed.

97231 The *LC_CTYPE* environment variable description is updated to align with the IEEE P1003.2b
97232 draft standard.

97233 The normative text is reworded to avoid use of the term ``must'' for application requirements.

97234 **Issue 7**

97235 Austin Group Interpretation 1003.1-2001 #027 is applied.

97236 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

97237 The *expand* utility is moved from the User Portability Utilities option to the Base. User
97238 Portability Utilities is now an option for interactive utilities.

97239 **Issue 8**

97240 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

97241 **NAME**

97242 expr — evaluate arguments as an expression

97243 **SYNOPSIS**

97244 `expr` *operand*`...`

97245 **DESCRIPTION**

97246 The *expr* utility shall evaluate an expression and write the result to standard output.

97247 **OPTIONS**

97248 None.

97249 **OPERANDS**

97250 The single expression evaluated by *expr* shall be formed from the *operand* operands, as described
97251 in the EXTENDED DESCRIPTION section. The application shall ensure that each of the
97252 expression operator symbols:

97253 `( ) | & = > >= < <= != + - * / % :`

97254 and the symbols *integer* and *string* in the table are provided as separate arguments to *expr*.

97255 **STDIN**

97256 Not used.

97257 **INPUT FILES**

97258 None.

97259 **ENVIRONMENT VARIABLES**

97260 The following environment variables shall affect the execution of *expr*:

97261 *LANG* Provide a default value for the internationalization variables that are unset or null.
97262 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
97263 variables used to determine the values of locale categories.)

97264 *LC_ALL* If set to a non-empty string value, override the values of all the other
97265 internationalization variables.

97266 *LC_COLLATE*
97267 Determine the locale for the behavior of ranges, equivalence classes, and multi-
97268 character collating elements within regular expressions and by the string
97269 comparison operators.

97270 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
97271 characters (for example, single-byte as opposed to multi-byte characters in
97272 arguments) and the behavior of character classes within regular expressions.

97273 *LC_MESSAGES*
97274 Determine the locale that should be used to affect the format and contents of
97275 diagnostic messages written to standard error.

97276 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

97277 **ASYNCHRONOUS EVENTS**

97278 Default.

97279 **STDOUT**

97280 The *expr* utility shall evaluate the expression and write the result, followed by a <newline>, to
97281 standard output.

97282 **STDERR**

97283 The standard error shall be used only for diagnostic messages.

97284 **OUTPUT FILES**

97285 None.

97286 **EXTENDED DESCRIPTION**

97287 The formation of the expression to be evaluated is shown in the following table. The symbols
97288 *expr*, *expr1*, and *expr2* represent expressions formed from *integer* and *string* symbols and the
97289 expression operator symbols (all separate arguments) by recursive application of the constructs
97290 described in the table. The expressions are listed in order of decreasing precedence, with equal-
97291 precedence operators grouped between horizontal lines. All of the operators shall be left-
97292 associative.

97293

Expression	Description
integer	An argument consisting only of an (optional) unary minus followed by digits.
string	A string argument; see below.
(*expr*)	Grouping symbols. Any expression can be placed within parentheses. Parentheses can be nested to a depth of {EXPR_NEST_MAX}.
expr1 : *expr2*	Matching expression; see below.
expr1 * *expr2*	Multiplication of decimal integer-valued arguments.
expr1 / *expr2*	Integer division of decimal integer-valued arguments, producing an integer result.
expr1 % *expr2*	Remainder of integer division of decimal integer-valued arguments.
expr1 + *expr2*	Addition of decimal integer-valued arguments.
expr1 – *expr2*	Subtraction of decimal integer-valued arguments.
	Returns the result of a decimal integer comparison if both arguments are integers; otherwise, returns the result of a string comparison using the locale-specific collation sequence. The result of each comparison is 1 if the specified relationship is true, or 0 if the relationship is false.
expr1 = *expr2*	Equal.
expr1 > *expr2*	Greater than.
expr1 >= *expr2*	Greater than or equal.
expr1 < *expr2*	Less than.
expr1 <= *expr2*	Less than or equal.
expr1 != *expr2*	Not equal.
expr1 & *expr2*	Returns the evaluation of *expr1* if neither expression evaluates to null or zero; otherwise, returns zero.
expr1 | *expr2*	Returns the evaluation of *expr1* if it is neither null nor zero; otherwise, returns the evaluation of *expr2* if it is not null; otherwise, zero.

97324 **Matching Expression**

97325 The `':'` matching operator shall compare the string resulting from the evaluation of *expr1* with
97326 the regular expression pattern resulting from the evaluation of *expr2*. Regular expression syntax
97327 shall be that defined in XBD Section 9.3 (on page 181), except that all patterns are anchored to
97328 the beginning of the string (that is, only sequences starting at the first character of a string are
97329 matched by the regular expression) and, therefore, it is unspecified whether `'^'` is a special
97330 character in that context. Usually, the matching operator shall return a string representing the
97331 number of characters matched (`'0'` on failure). Alternatively, if the pattern contains at least one
97332 regular expression subexpression `"\(...\)"`, the string matched by the back-reference
97333 expression `"\1"` shall be returned. If the back-reference expression `"\1"` does not match, then
97334 the null string shall be returned.

97335 **Identification as Integer or String**

97336 An argument or the value of a subexpression that consists only of an optional unary minus
97337 followed by digits is a candidate for treatment as an integer if it is used as the left argument to
97338 the `|` operator or as either argument to any of the following operators: `&` `=` `>` `>=` `<` `<=` `!=` `+`
97339 `-` `*` `/` `%`. Otherwise, the argument or subexpression value shall be treated as a string.

97340 The use of string arguments **length**, **substr**, **index**, or **match** produces unspecified results.

97341 **EXIT STATUS**

97342 The following exit values shall be returned:

97343 0 The *expression* evaluated to neither null nor zero, and the output specified in STDOUT was
97344 successfully written to standard output.

97345 1 The *expression* evaluated to null or zero, and the output specified in STDOUT was
97346 successfully written to standard output.

97347 2 Invalid *expression* error.

97348 >2 Another error occurred.

97349 **CONSEQUENCES OF ERRORS**

97350 Default.

97351 **APPLICATION USAGE**

97352 The *expr* utility has a rather difficult syntax:

97353 • Many of the operators are also shell control operators or reserved words, so they have to
97354 be escaped on the command line.

97355 • Each part of the expression is composed of separate arguments, so liberal usage of <blank>
97356 characters is required. For example:

97357

Invalid	Valid
`expr 1+2`	`expr 1 + 2`
`expr "1 + 2"`	`expr 1 + 2`
`expr 1 + (2 * 3)`	`expr 1 + \( 2 \* 3 \)`

97358
97359
97360

97361 In many cases, the arithmetic and string features provided as part of the shell command
97362 language are easier to use than their equivalents in *expr*. Newly written scripts should avoid
97363 *expr* in favor of the new features within the shell; see Section 2.5 (on page 2478) and Section 2.6.4
97364 (on page 2490).

97365 After argument processing by the shell, *expr* is not required to be able to tell the difference
97366 between an operator and an operand except by the value. If `"$a"` is `'='`, the command:


```
expr "$a" = '='
```

looks like:

```
expr = = =
```

as the arguments are passed to *expr* (and they all may be taken as the `'='` operator). The following works reliably:

```
expr "X$a" = X=
```

Also note that this volume of POSIX.1-2024 permits implementations to extend utilities. The *expr* utility permits the integer arguments to be preceded with a unary minus. This means that an integer argument could look like an option. Therefore, the conforming application must employ the `"--"` construct of Guideline 10 of XBD Section 12.2 (on page 215) to protect its operands if there is any chance the first operand might be a negative integer (or any string with a leading minus).

For testing string equality the *test* utility is preferred over *expr*, as it is usually implemented as a shell built-in. However, the functionality is not quite the same because the *expr* `=` and `!=` operators check whether strings collate equally, whereas *test* checks whether they are identical. Therefore, they can produce different results in locales where the collation sequence does not have a total ordering of all characters (see XBD Section 7.3.2, on page 139).

EXAMPLES

The following command:

```
a=$(expr "$a" + 1)
```

adds 1 to the variable *a*.

The following command, for `"$a"` equal to either **/usr/abc/file** or just **file**:

```
expr $a : '.*/\(.*\)' \| $a
```

returns the last segment of a pathname (that is, **file**). Applications should avoid the character `'/'` used alone as an argument; *expr* may interpret it as the division operator.

The following command:

```
expr "//$a" : '.*/\(.*\)'
```

is a better representation of the previous example. The addition of the `"//"` characters eliminates any ambiguity about the division operator and simplifies the whole expression. Also note that pathnames may contain characters contained in the *IFS* variable and should be quoted to avoid having `"$a"` expand into multiple arguments.

The following command:

```
expr "X$VAR" : '.*' - 1
```

returns the number of characters in *VAR*.

RATIONALE

In an early proposal, EREs were used in the matching expression syntax. This was changed to BREs to avoid breaking historical applications.

The use of a leading <circumflex> in the BRE is unspecified because many historical implementations have treated it as a special character, despite their system documentation. For example:

```
expr foo : ^foo     expr ^foo : ^foo
```


return 3 and 0, respectively, on those systems; their documentation would imply the reverse. Thus, the anchoring condition is left unspecified to avoid breaking historical scripts relying on this undocumented feature.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 2478), Section 2.6.4 (on page 2490)

XBD Section 7.3.2 (on page 139), Chapter 8 (on page 167), Section 9.3 (on page 181), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

The *expr* utility is aligned with the IEEE P1003.2b draft standard, to include resolution of IEEE PASC Interpretation 1003.2 #104.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.

The SYNOPSIS and OPERANDS sections are revised to explicitly state that the name of each of the operands is *operand*.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0094 [942], XCU/TC2-2008/0095 [709], XCU/TC2-2008/0096 [942], XCU/TC2-2008/0097 [963], and XCU/TC2-2008/0098 [942] are applied.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1500 is applied, changing the EXIT STATUS section.

Austin Group Defect 1757 is applied, changing `"[\(...\)]"` to `"\(...\)"`.

NAME

false — return false value

SYNOPSIS

```
false
```

DESCRIPTION

The *false* utility shall return with a non-zero exit code.

OPTIONS

None.

OPERANDS

None.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

Not used.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The *false* utility shall always exit with a value between 1 and 125, inclusive.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

true

97476 **CHANGE HISTORY**

97477 First released in Issue 2.

97478 **Issue 6**

97479 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/24 is applied, changing the STDERR
97480 section from ``None.'' to ``Not used.'' for alignment withSection 1.4 (on page 2462).

97481 **Issue 8**

97482 Austin Group Defect 1321 is applied, changing the EXIT STATUS section.

NAME

fc — process the command history list

SYNOPSIS

UP

```
fc [-r] [-e editor] [first [last]]
fc -l [-nr] [first [last]]
fc -s [old=new] [first]
```

DESCRIPTION

The *fc* utility shall list, or shall edit and re-execute, commands previously entered to an interactive *sh*.

The command history list shall reference commands by number. The first number in the list is selected arbitrarily. The relationship of a number to its command shall not change except when the user logs in and no other process is accessing the list, at which time the system may reset the numbering to start the oldest retained command at another number (usually 1). When the number reaches an implementation-defined upper limit, which shall be no smaller than the value in *HISTSIZE* or 32 767 (whichever is greater), the shell may wrap the numbers, starting the next command with a lower number (usually 1). However, despite this optional wrapping of numbers, *fc* shall maintain the time-ordering sequence of the commands. For example, if four commands in sequence are given the numbers 32 766, 32 767, 1 (wrapped), and 2 as they are executed, command 32 767 is considered the command previous to 1, even though its number is higher.

When commands are edited (when the **–l** option is not specified), the resulting lines shall be entered at the end of the history list and then re-executed by *sh*. The *fc* command that caused the editing shall not be entered into the history list. If the editor returns a non-zero exit status, this shall suppress the entry into the history list and the command re-execution. Any command line variable assignments or redirection operators used with *fc* shall affect both the *fc* command itself as well as the command that results; for example:

```
fc -s -- -1 2>/dev/null
```

reinvokes the previous command, suppressing standard error for both *fc* and the previous command.

OPTIONS

The *fc* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–e *editor* Use the editor named by *editor* to edit the commands. The *editor* string is a utility name, subject to search via the *PATH* variable (see XBD Chapter 8, on page 167). The value in the *FCEDIT* variable shall be used as a default when **–e** is not specified. If *FCEDIT* is null or unset, *ed* shall be used as the editor.

–l (The letter ell.) List the commands rather than invoking an editor on them. The commands shall be written in the sequence indicated by the *first* and *last* operands, as affected by **–r**, with each command preceded by the command number.

–n Suppress command numbers when listing with **–l**.

–r Reverse the order of the commands listed (with **–l**) or edited (with neither **–l** nor **–s**).

97525 **–s** Re-execute the command without invoking an editor.

97526 **OPERANDS**

97527 The following operands shall be supported:

97528 *first*, *last* Select the commands to list or edit. The number of previous commands that can be
97529 accessed shall be determined by the value of the *HISTSIZE* variable. The value of
97530 *first* or *last* or both shall be one of the following:

97531 **[+]***number* A positive number representing a command number; command
97532 numbers can be displayed with the **–l** option.

97533 *–number* A negative decimal number representing the command that was
97534 executed *number* of commands previously. For example, –1 is the
97535 immediately previous command.

97536 *string* A string indicating the most recently entered command that begins
97537 with that string. If the *old=new* operand is not also specified with **–s**,
97538 the string form of the *first* operand cannot contain an embedded
97539 <equals-sign>.

97540 When the synopsis form with **–s** is used:

97541 • If *first* is omitted, the previous command shall be used.

97542 For the synopsis forms without **–s**:

97543 • If *last* is omitted, *last* shall default to the previous command when **–l** is
97544 specified; otherwise, it shall default to *first*.

97545 • If *first* and *last* are both omitted, the previous 16 commands shall be listed or
97546 the previous single command shall be edited (based on the **–l** option).

97547 • If *first* and *last* are both present, all of the commands from *first* to *last* shall be
97548 edited (without **–l**) or listed (with **–l**). Editing multiple commands shall be
97549 accomplished by presenting to the editor all of the commands at one time,
97550 each command starting on a new line. If *first* represents a newer command
97551 than *last*, the commands shall be listed or edited in reverse sequence,
97552 equivalent to using **–r**. For example, the following commands on the first
97553 line are equivalent to the corresponding commands on the second:

97554 `fc -r 10 20    fc    30 40`
97555 `fc    20 10    fc -r 40 30`

97556 • When a range of commands is used, it shall not be an error to specify *first* or
97557 *last* values that are not in the history list; *fc* shall substitute the value
97558 representing the oldest or newest command in the list, as appropriate. For
97559 example, if there are only ten commands in the history list, numbered 1 to 10:

97560 `fc -l`
97561 `fc 1 99`

97562 shall list and edit, respectively, all ten commands.

97563 *old=new* Replace the first occurrence of string *old* in the commands to be re-executed by the
97564 string *new*.

97565 **STDIN**

97566 Not used.

97567 **INPUT FILES**

97568 None.

97569 **ENVIRONMENT VARIABLES**

97570 The following environment variables shall affect the execution of *fc*:

97571 *FCEDIT* This variable, when expanded by the shell, shall determine the default value for
97572 the **−e** *editor* option's *editor* option-argument. If *FCEDIT* is null or unset, *ed* shall be
97573 used as the editor.

97574 *HISTFILE* Determine a pathname naming a command history file. If the *HISTFILE* variable is
97575 not set, the shell may attempt to access or create a file **.sh_history** in the directory
97576 referred to by the *HOME* environment variable. If the shell cannot obtain both read
97577 and write access to, or create, the history file, it shall use an unspecified
97578 mechanism that allows the history to operate properly. (References to history ``file''
97579 in this section shall be understood to mean this unspecified mechanism in such
97580 cases.) An implementation may choose to access this variable only when
97581 initializing the history file; this initialization shall occur when *fc* or *sh* first attempt
97582 to retrieve entries from, or add entries to, the file, as the result of commands issued
97583 by the user, the file named by the *ENV* variable, or implementation-defined system
97584 start-up files. In some historical shells, the history file is initialized just after the
97585 *ENV* file has been processed. Therefore, it is implementation-defined whether
97586 changes made to *HISTFILE* after the history file has been initialized are effective.
97587 Implementations may choose to disable the history list mechanism for users with
97588 appropriate privileges who do not set *HISTFILE*; the specific circumstances under
97589 which this occurs are implementation-defined. If more than one instance of the
97590 shell is using the same history file, it is unspecified how updates to the history file
97591 from those shells interact. As entries are deleted from the history file, they shall be
97592 deleted oldest first. It is unspecified when history file entries are physically
97593 removed from the history file.

97594 *HISTSIZE* Determine a decimal number representing the limit to the number of previous
97595 commands that are accessible. If this variable is unset, an unspecified default
97596 greater than or equal to 128 shall be used. The maximum number of commands in
97597 the history list is unspecified, but shall be at least 128. An implementation may
97598 choose to access this variable only when initializing the history file, as described
97599 under *HISTFILE*. Therefore, it is unspecified whether changes made to *HISTSIZE*
97600 after the history file has been initialized are effective.

97601 *LANG* Provide a default value for the internationalization variables that are unset or null.
97602 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
97603 variables used to determine the values of locale categories.)

97604 *LC_ALL* If set to a non-empty string value, override the values of all the other
97605 internationalization variables.

97606 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
97607 characters (for example, single-byte as opposed to multi-byte characters in
97608 arguments and input files).

97609 *LC_MESSAGES*

97610 Determine the locale that should be used to affect the format and contents of
97611 diagnostic messages written to standard error.

97612 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

97613 **ASYNCHRONOUS EVENTS**

97614 Default.

97615 **STDOUT**

97616 When the **–l** option is used to list commands, the format of each command in the list shall be as
97617 follows:

97618 `"%d\t%s\n", <line number>, <command>`

97619 If both the **–l** and **–n** options are specified, the format of each command shall be:

97620 `"\t%s\n", <command>`

97621 If the **<***command***>** consists of more than one line, the lines after the first shall be displayed as:

97622 `"\t%s\n", <continued-command>`

97623 **STDERR**

97624 The standard error shall be used only for diagnostic messages.

97625 **OUTPUT FILES**

97626 None.

97627 **EXTENDED DESCRIPTION**

97628 None.

97629 **EXIT STATUS**

97630 The following exit values shall be returned:

97631 0 Successful completion of the listing.

97632 **>**0 An error occurred.

97633 Otherwise, the exit status shall be that of the commands executed by *fc*.

97634 **CONSEQUENCES OF ERRORS**

97635 Default.

97636 **APPLICATION USAGE**

97637 This utility is required to be intrinsic. See Section 1.7 (on page 2470) for details.

97638 Since editors sometimes use file descriptors as integral parts of their editing, redirecting their file
97639 descriptors as part of the *fc* command can produce unexpected results. For example, if *vi* is the
97640 *FCEDIT* editor, the command:

97641 `fc -s | more`

97642 does not work correctly on many systems.

97643 Users on windowing systems may want to have separate history files for each window by
97644 setting *HISTFILE* as follows:

97645 `HISTFILE=$HOME/.sh_hist$$`

97646 **EXAMPLES**

97647 None.

97648 **RATIONALE**

97649 This utility is based on the *fc* built-in of the KornShell.

97650 An early proposal specified the **–e** option as **[–e** *editor* **[***old***=** *new* **]]**, which is not historical
97651 practice. Historical practice in *fc* of either **[–e** *editor***]** or **[–e** – **[** *old***=** *new* **]]** is acceptable, but not

both together. To clarify this, a new option **–s** was introduced replacing the **[–e –]**. This resolves the conflict and makes *fc* conform to the Utility Syntax Guidelines.

HISTFILE Some implementations of the KornShell check for the superuser and do not create a history file unless *HISTFILE* is set. This is done primarily to avoid creating unlinked files in the root file system when logging in during single-user mode. *HISTFILE* must be set for the superuser to have history.

HISTSIZE Needed to limit the size of history files. It is the intent of the standard developers that when two shells share the same history file, commands that are entered in one shell shall be accessible by the other shell. Because of the difficulties of synchronization over a network, the exact nature of the interaction is unspecified.

The initialization process for the history file can be dependent on the system start-up files, in that they may contain commands that effectively preempt the settings the user has for *HISTFILE* and *HISTSIZE*. For example, function definition commands are recorded in the history file. If the system administrator includes function definitions in some system start-up file called before the *ENV* file, the history file is initialized before the user can influence its characteristics. In some historical shells, the history file is initialized just after the *ENV* file has been processed. Because of these situations, the text requires the initialization process to be implementation-defined.

Consideration was given to omitting the *fc* utility in favor of the command line editing feature in *sh*. For example, in *vi* editing mode, typing `"<ESC> v"` is equivalent to:

```
EDITOR=vi fc
```

However, the *fc* utility allows the user the flexibility to edit multiple commands simultaneously (such as *fc* 10 20) and to use editors other than those supported by *sh* for command line editing.

In the KornShell, the alias **r** (``re-do'') is preset to *fc* **–e** – (equivalent to the POSIX *fc* **–s**). This is probably an easier command name to remember than *fc* (``fix command''), but it does not meet the Utility Syntax Guidelines. Renaming *fc* to *hist* or *redo* was considered, but since this description closely matches historical KornShell practice already, such a renaming was seen as gratuitous. Users are free to create aliases whenever odd historical names such as *fc*, *awk*, *cat*, *grep*, or *yacc* are standardized by POSIX.

Command numbers have no ordering effects; they are like serial numbers. The **–r** option and *–number* operand address the sequence of command execution, regardless of serial numbers. So, for example, if the command number wrapped back to 1 at some arbitrary point, there would be no ambiguity associated with traversing the wrap point. For example, if the command history were:

```
32766: echo 1
32767: echo 2
1: echo 3
```

the number –2 refers to command 32 767 because it is the second previous command, regardless of serial number.

FUTURE DIRECTIONS

None.

SEE ALSO

sh

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

97695 **CHANGE HISTORY**

97696 First released in Issue 4.

97697 **Issue 5**

97698 The FUTURE DIRECTIONS section is added.

97699 **Issue 6**

97700 This utility is marked as part of the User Portability Utilities option.

97701 In the ENVIRONMENT VARIABLES section, the text ``user's home directory'' is updated to
97702 ``directory referred to by the *HOME* environment variable''.

97703 **Issue 7**

97704 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

97705 **Issue 8**

97706 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that
97707 this utility is required to be intrinsic.

97708 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

97709 **NAME**

97710 fg — run jobs in the foreground

97711 **SYNOPSIS**

97712 UP `fg [job_id]`

97713 **DESCRIPTION**

97714 If job control is enabled (see the description of *set* −**m**), the shell is interactive, and the current
97715 shell execution environment (see Section 2.13, on page 2522) is not a subshell environment, the *fg*
97716 utility shall move a background job in the current execution environment into the foreground, as
97717 described in Section 2.11 (on page 2518); it may also do so if the shell is non-interactive or the
97718 current shell execution environment is a subshell environment.

97719 Using *fg* to place a job into the foreground shall remove its process ID from the list of those
97720 ``known in the current shell execution environment''; see Section 2.9.3.1 (on page 2506).

97721 **OPTIONS**

97722 None.

97723 **OPERANDS**

97724 The following operand shall be supported:

97725 *job_id* Specify the job to be run as a foreground job. If no *job_id* operand is given, the
97726 *job_id* for the job that was most recently suspended, placed in the background, or
97727 run as a background job shall be used. The format of *job_id* is described in XBD
97728 Section 3.182 (on page 57).

97729 **STDIN**

97730 Not used.

97731 **INPUT FILES**

97732 None.

97733 **ENVIRONMENT VARIABLES**

97734 The following environment variables shall affect the execution of *fg*:

97735 *LANG* Provide a default value for the internationalization variables that are unset or null.
97736 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
97737 variables used to determine the values of locale categories.)

97738 *LC_ALL* If set to a non-empty string value, override the values of all the other
97739 internationalization variables.

97740 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
97741 characters (for example, single-byte as opposed to multi-byte characters in
97742 arguments).

97743 *LC_MESSAGES*
97744 Determine the locale that should be used to affect the format and contents of
97745 diagnostic messages written to standard error.

97746 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

97747 **ASYNCHRONOUS EVENTS**

97748 Default.

97749 **STDOUT**

97750 The *fg* utility shall write the command line of the job to standard output in the following format:

97751 `"%s\n", <command>`

97752 **STDERR**

97753 The standard error shall be used only for diagnostic messages.

97754 **OUTPUT FILES**

97755 None.

97756 **EXTENDED DESCRIPTION**

97757 None.

97758 **EXIT STATUS**

97759 If the *fg* utility succeeds, it does not return an exit status. Instead, the shell waits for the job that
97760 *fg* moved into the foreground.

97761 If *fg* does not move a job into the foreground, the following exit value shall be returned:

97762 >0 An error occurred.

97763 **CONSEQUENCES OF ERRORS**

97764 If job control is disabled, the *fg* utility shall exit with an error and no job shall be placed in the
97765 foreground.

97766 **APPLICATION USAGE**

97767 This utility is required to be intrinsic. See Section 1.7 (on page 2470) for details.

97768 The *fg* utility does not work as expected when it is operating in its own utility execution
97769 environment because that environment has no applicable jobs to manipulate. See the
97770 APPLICATION USAGE section for *bg*. For this reason, *fg* is generally implemented as a shell
97771 regular built-in.

97772 **EXAMPLES**

97773 None.

97774 **RATIONALE**

97775 The extensions to the shell specified in this volume of POSIX.1-2024 have mostly been based on
97776 features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs* are also
97777 based on the KornShell. The standard developers examined the characteristics of the C shell
97778 versions of these utilities and found that differences exist. Despite widespread use of the C shell,
97779 the KornShell versions were selected for this volume of POSIX.1-2024 to maintain a degree of
97780 uniformity with the rest of the KornShell features selected (such as the very popular command
97781 line editing features).

97782 **FUTURE DIRECTIONS**

97783 None.

97784 **SEE ALSO**

97785 Section 2.9.3.1 (on page 2506), Section 2.13 (on page 2522), *bg*, *kill*, *jobs*, *wait*

97786 XBD Section 3.182 (on page 57), Chapter 8 (on page 167)

97787 **CHANGE HISTORY**

97788 First released in Issue 4.

97789 **Issue 6**

97790 This utility is marked as part of the User Portability Utilities option.

97791 The APPLICATION USAGE section is added.

97792 The JC marking is removed from the SYNOPSIS since job control is mandatory is this version.

97793 **Issue 8**

97794 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that
97795 this utility is required to be intrinsic.

97796 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

97797 Austin Group Defect 1254 is applied, updating the DESCRIPTION to account for the addition of
97798 Section 2.11 (on page 2518) and changing the EXIT STATUS section.

97799 **NAME**

97800 file — determine file type

97801 **SYNOPSIS**

97802 `file [-dh] [-M file] [-m file] file...`

97803 `file -i [-h] file...`

97804 **DESCRIPTION**

97805 The *file* utility shall perform a series of tests in sequence on each specified *file* in an attempt to
97806 classify it:

97807 1. If *file* does not exist, cannot be read, or its file status could not be determined, the output
97808 shall indicate that the file was processed, but that its type could not be determined.

97809 2. If the file is not a regular file, its file type shall be identified. The file types directory,
97810 FIFO, socket, block special, and character special shall be identified as such. Other
97811 implementation-defined file types may also be identified. If *file* is a symbolic link, by
97812 default the link shall be resolved and *file* shall test the type of file referenced by the
97813 symbolic link. (See the **–h** and **–i** options below.)

97814 3. If the length of *file* is zero, it shall be identified as an empty file.

97815 4. The *file* utility shall examine an initial segment of *file* and shall make a guess at
97816 identifying its contents based on position-sensitive tests. (The answer is not guaranteed to
97817 be correct; see the **–d**, **–M**, and **–m** options below.)

97818 5. The *file* utility shall examine *file* and make a guess at identifying its contents based on
97819 context-sensitive default system tests. (The answer is not guaranteed to be correct.)

97820 6. The file shall be identified as a data file.

97821 If *file* does not exist, cannot be read, or its file status could not be determined, the output shall
97822 indicate that the file was processed, but that its type could not be determined.

97823 If *file* is a symbolic link, by default the link shall be resolved and *file* shall test the type of file
97824 referenced by the symbolic link.

97825 **OPTIONS**

97826 The *file* utility shall conform to XBD Section 12.2 (on page 215), except that the order of the **–m**,
97827 **–d**, and **–M** options shall be significant.

97828 The following options shall be supported by the implementation:

97829 **–d** Apply any position-sensitive default system tests and context-sensitive default
97830 system tests to the file. This is the default if no **–M** or **–m** option is specified.

97831 **–h** When a symbolic link is encountered, identify the file as a symbolic link. If **–h** is
97832 not specified and *file* is a symbolic link that refers to a nonexistent file, *file* shall
97833 identify the file as a symbolic link, as if **–h** had been specified.

97834 **–i** If a file is a regular file, do not attempt to classify the type of the file further, but
97835 identify the file as specified in the STDOUT section.

97836 **–M** *file* Specify the name of a file containing position-sensitive tests that shall be applied to
97837 a file in order to classify it (see the EXTENDED DESCRIPTION). No position-
97838 sensitive default system tests nor context-sensitive default system tests shall be
97839 applied unless the **–d** option is also specified.

97840 –**m** *file* Specify the name of a file containing position-sensitive tests that shall be applied to
97841 a file in order to classify it (see the EXTENDED DESCRIPTION).

97842 If the –**m** option is specified without specifying the –**d** option or the –**M** option, position-
97843 sensitive default system tests shall be applied after the position-sensitive tests specified by the
97844 –**m** option. If the –**M** option is specified with the –**d** option, the –**m** option, or both, or the –**m**
97845 option is specified with the –**d** option, the concatenation of the position-sensitive tests specified
97846 by these options shall be applied in the order specified by the appearance of these options. If a
97847 –**M** or –**m** *file* option-argument is –, the results are unspecified.

97848 **OPERANDS**

97849 The following operand shall be supported:

97850 *file* A pathname of a file to be tested.

97851 **STDIN**

97852 The standard input shall be used if a *file* operand is `'–'` and the implementation treats the `'–'`
97853 as meaning standard input. Otherwise, the standard input shall not be used.

97854 **INPUT FILES**

97855 The *file* can be any file type.

97856 **ENVIRONMENT VARIABLES**

97857 The following environment variables shall affect the execution of *file*:

97858 *LANG* Provide a default value for the internationalization variables that are unset or null.
97859 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
97860 variables used to determine the values of locale categories.)

97861 *LC_ALL* If set to a non-empty string value, override the values of all the other
97862 internationalization variables.

97863 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
97864 characters (for example, single-byte as opposed to multi-byte characters in
97865 arguments and input files).

97866 *LC_MESSAGES*
97867 Determine the locale that should be used to affect the format and contents of
97868 diagnostic messages written to standard error and informative messages written to
97869 standard output.

97870 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

97871 **ASYNCHRONOUS EVENTS**

97872 Default.

97873 **STDOUT**

97874 In the POSIX locale, the following format shall be used to identify each operand, *file* specified:

97875 `"%s: %s\n", <file>, <type>`

97876 The values for <*type*> are unspecified, except that in the POSIX locale, if *file* is identified as one
97877 of the types listed in the following table, <*type*> shall contain (but is not limited to) the
97878 corresponding string, unless the file is identified by a position-sensitive test specified by a –**M** or
97879 –**m** option. Each <space> shown in the strings shall be exactly one <space>.

Table 3-10 File Utility Output Strings

If *file* is:	<*type*> shall contain the string:	Notes
Nonexistent	cannot open	
Block special	block special	1
Character special	character special	1
Directory	directory	1
FIFO	fifo	1
Socket	socket	1
Symbolic link	symbolic link to	1
Regular file	regular file	1,2
Empty regular file	empty	3
Regular file that cannot be read	cannot open	3
Executable binary	executable	3,4,6
ar archive library (see *ar*)	archive	3,4,6
Extended *cpio* format (see *pax*)	cpio archive	3,4,6
Extended *tar* format (see **ustar** in *pax*)	tar archive	3,4,6
Shell script	commands text	3,5,6
C-language source	c program text	3,5,6
FORTRAN source	fortran program text	3,5,6
Regular file whose type cannot be determined	data	3

Notes:

1. This is a file type test.
2. This test is applied only if the **–i** option is specified.
3. This test is applied only if the **–i** option is not specified.
4. This is a position-sensitive default system test.
5. This is a context-sensitive default system test.
6. Position-sensitive default system tests and context-sensitive default system tests are not applied if the **–M** option is specified unless the **–d** option is also specified.

In the POSIX locale, if *file* is identified as a symbolic link (see the **–h** option), the following alternative output format shall be used:

```
"%s: %s %s\n", <file>, <type>, <contents of link>"
```

If the file named by the *file* operand does not exist, cannot be read, or the type of the file named by the *file* operand cannot be determined, this shall not be considered an error that affects the exit status.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

97918 **EXTENDED DESCRIPTION**

97919 A file specified as an option-argument to the **–m** or **–M** options shall contain one position-
97920 sensitive test per line, which shall be applied to the file. If the test succeeds, the message field of
97921 the line shall be printed and no further tests shall be applied, with the exception that tests on
97922 immediately following lines beginning with a single `'>'` character shall be applied.

97923 Each line shall be composed of the following four **<**tab**>**-separated fields. (Implementations may
97924 allow any combination of one or more white-space characters other than **<**newline**>** to act as
97925 field separators.)

97926 *offset* An unsigned number (optionally preceded by a single `'>'` character) specifying
97927 the *offset*, in bytes, of the value in the file that is to be compared against the *value*
97928 field of the line. If the file is shorter than the specified offset, the test shall fail.

97929 If the *offset* begins with the character `'>'`, the test contained in the line shall not be
97930 applied to the file unless the test on the last line for which the *offset* did not begin
97931 with a `'>'` was successful. By default, the *offset* shall be interpreted as an unsigned
97932 decimal number. With a leading 0x or 0X, the *offset* shall be interpreted as a
97933 hexadecimal number; otherwise, with a leading 0, the *offset* shall be interpreted as
97934 an octal number.

97935 *type* The type of the value in the file to be tested. The type shall consist of the type
97936 specification characters `d`, `s`, and `u`, specifying signed decimal, string, and
97937 unsigned decimal, respectively.

97938 The *type* string shall be interpreted as the bytes from the file starting at the
97939 specified *offset* and including the same number of bytes specified by the *value* field.
97940 If insufficient bytes remain in the file past the *offset* to match the *value* field, the test
97941 shall fail.

97942 The type specification characters `d` and `u` can be followed by an optional unsigned
97943 decimal integer that specifies the number of bytes represented by the type. The
97944 type specification characters `d` and `u` can be followed by an optional `C`, `S`, `I`, or `L`,
97945 indicating that the value is of type **char**, **short**, **int**, or **long**, respectively.

97946 The default number of bytes represented by the type specifiers `d`, `f`, and `u` shall
97947 correspond to their respective C-language types as follows. If the system claims
97948 conformance to the C-Language Development Utilities option, those specifiers
97949 shall correspond to the default sizes used in the *c17* utility. Otherwise, the default
97950 sizes shall be implementation-defined.

97951 For the type specifier characters `d` and `u`, the default number of bytes shall
97952 correspond to the size of a basic integer type of the implementation. For these
97953 specifier characters, the implementation shall support values of the optional
97954 number of bytes to be converted corresponding to the number of bytes in the C-
97955 language types **char**, **short**, **int**, or **long**. These numbers can also be specified by an
97956 application as the characters `C`, `S`, `I`, and `L`, respectively. The byte order used when
97957 interpreting numeric values is implementation-defined, but shall correspond to the
97958 order in which a constant of the corresponding type is stored in memory on the
97959 system.

97960 All type specifiers, except for `s`, can be followed by a mask specifier of the form
97961 &*number*. The mask value shall be AND'ed with the value of the input file before
97962 the comparison with the *value* field of the line is made. By default, the mask shall
97963 be interpreted as an unsigned decimal number. With a leading 0x or 0X, the mask
97964 shall be interpreted as an unsigned hexadecimal number; otherwise, with a leading

97965 0, the mask shall be interpreted as an unsigned octal number.

97966 The strings **byte**, **short**, **long**, and **string** shall also be supported as type fields,
97967 being interpreted as `dC`, `dS`, `dL`, and `s`, respectively.

97968 *value* The *value* to be compared with the value from the file.

97969 If the specifier from the type field is `s` or **string**, then interpret the value as a string.
97970 Otherwise, interpret it as a number. If the value is a string, then the test shall
97971 succeed only when a string value exactly matches the bytes from the file.

97972 If the *value* is a string, it can contain the following sequences:

97973 *character* The <backslash>-escape sequences as specified in XBD Table 5-1
97974 (on page 113) (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`,
97975 `'\v'`). In addition, the escape sequence `'\ '` (the <backslash>
97976 character followed by a <space> character) shall be recognized to
97977 represent a <space> character. The results of using any other
97978 character, other than an octal digit, following the <backslash>
97979 are unspecified.

97980 *octal* Octal sequences that can be used to represent characters with
97981 specific coded values. An octal sequence shall consist of a
97982 <backslash> followed by the longest sequence of one, two, or
97983 three octal-digit characters (01234567).

97984 By default, any value that is not a string shall be interpreted as a signed decimal
97985 number. Any such value, with a leading 0x or 0X, shall be interpreted as an
97986 unsigned hexadecimal number; otherwise, with a leading zero, the value shall be
97987 interpreted as an unsigned octal number.

97988 If the value is not a string, it can be preceded by a character indicating the
97989 comparison to be performed. Permissible characters and the comparisons they
97990 specify are as follows:

97991 = The test shall succeed if the value from the file equals the *value* field.

97992 < The test shall succeed if the value from the file is less than the *value* field.

97993 > The test shall succeed if the value from the file is greater than the *value* field.

97994 & The test shall succeed if all of the set bits in the *value* field are set in the value
97995 from the file.

97996 ^ The test shall succeed if at least one of the set bits in the *value* field is not set in
97997 the value from the file.

97998 x The test shall succeed if the file is large enough to contain a value of the type
97999 specified starting at the offset specified.

98000 *message* The *message* to be printed if the test succeeds. The *message* shall be interpreted
98001 using the notation for the *printf* formatting specification; see *printf*. If the *value*
98002 field was a string, then the value from the file shall be the argument for the *printf*
98003 formatting specification; otherwise, the value from the file shall be the argument.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *file* utility can only be required to guess at many of the file types because only exhaustive testing can determine some types with certainty. For example, binary data on some implementations might match the initial segment of an executable or a *tar* archive.

Note that the table indicates that the output contains the stated string. Systems may add text before or after the string. For executables, as an example, the machine architecture and various facts about how the file was link-edited may be included. Note also that on systems that recognize shell script files starting with `"#!"` as executable files, these may be identified as executable binary files rather than as shell scripts.

EXAMPLES

Determine whether an argument is a binary executable file:

```
file -- "$1" | grep -q ':.*executable' &&
    printf "%s is executable.\n" "$1"
```

RATIONALE

The **–f** option was omitted because the same effect can (and should) be obtained using the *xargs* utility.

Historical versions of the *file* utility attempt to identify the following types of files: symbolic link, directory, character special, block special, socket, *tar* archive, *cpio* archive, SCCS archive, archive library, empty, *compress* output, *pack* output, binary data, C source, FORTRAN source, assembler source, *nroff/troff/eqn/tbl* source *troff* output, shell script, C shell script, English text, ASCII text, various executables, APL workspace, compiled terminfo entries, and CURSES screen images. Only those types that are reasonably well specified in POSIX or are directly related to POSIX utilities are listed in the table.

Historical systems have used a ``magic file'' named **/etc/magic** to help identify file types. Because it is generally useful for users and scripts to be able to identify special file types, the **–m** flag and a portable format for user-created magic files has been specified. No requirement is made that an implementation of *file* use this method of identifying files, only that users be permitted to add their own classifying tests.

In addition, three options have been added to historical practice. The **–d** flag has been added to permit users to cause their tests to follow any default system tests. The **–i** flag has been added to permit users to test portably for regular files in shell scripts. The **–M** flag has been added to permit users to ignore any default system tests.

The POSIX.1-2024 description of default system tests and the interaction between the **–d**, **–M**, and **–m** options did not clearly indicate that there were two types of ``default system tests''. The ``position-sensitive tests'' determine file types by looking for certain string or binary values at specific offsets in the file being examined. These position-sensitive tests were implemented in historical systems using the magic file described above. Some of these tests are now built into the *file* utility itself on some implementations so the output can provide more detail than can be provided by magic files. For example, a magic file can easily identify a file containing a core image on most implementations, but cannot name the program file that dropped the core. A

98050 magic file could produce output such as:

98051 `/home/dwc/core: ELF 32-bit MSB core file SPARC Version 1`

98052 but by building the test into the *file* utility, you could get output such as:

98053 `/home/dwc/core: ELF 32-bit MSB core file SPARC Version 1, from 'testprog'`

98054 These extended built-in tests are still to be treated as position-sensitive default system tests even
98055 if they are not listed in **/etc/magic** or any other magic file.

98056 The context-sensitive default system tests were always built into the *file* utility. These tests
98057 looked for language constructs in text files trying to identify shell scripts, C, FORTRAN, and
98058 other computer language source files, and even plain text files. With the addition of the –**m** and
98059 –**M** options the distinction between position-sensitive and context-sensitive default system tests
98060 became important because the order of testing is important. The context-sensitive system default
98061 tests should never be applied before any position-sensitive tests even if the –**d** option is specified
98062 before a –**m** option or –**M** option due to the high probability that the context-sensitive system
98063 default tests will incorrectly identify arbitrary text files as text files before position-sensitive tests
98064 specified by the –**m** or –**M** option would be applied to give a more accurate identification.

98065 Leaving the meaning of –**M** – and –**m** – unspecified allows an existing prototype of these
98066 options to continue to work in a backwards-compatible manner. (In that implementation, –**M** –
98067 was roughly equivalent to –**d** in POSIX.1-2024.)

98068 The historical –**c** option was omitted as not particularly useful to users or portable shell scripts.
98069 In addition, a reasonable implementation of the *file* utility would report any errors found each
98070 time the magic file is read.

98071 The historical format of the magic file was the same as that specified by the Rationale in the
98072 ISO POSIX-2: 1993 standard for the *offset*, *value*, and *message* fields; however, it used less precise
98073 type fields than the format specified by the current normative text. The new type field values are
98074 a superset of the historical ones.

98075 The following is an example magic file:

```
98076 0  short     070707              cpio archive
98077 0  short     0143561             Byte-swapped cpio archive
98078 0  string    070707              ASCII cpio archive
98079 0  long      0177555             Very old archive
98080 0  short     0177545             Old archive
98081 0  short     017437              Old packed data
98082 0  string    \037\036            Packed data
98083 0  string    \377\037            Compacted data
98084 0  string    \037\235            Compressed data
98085 >2 byte&0x80 >0                  Block compressed
98086 >2 byte&0x1f x                   %d bits
98087 0  string    \032\001            Compiled Terminfo Entry
98088 0  short     0433                Curses screen image
98089 0  short     0434                Curses screen image
98090 0  string    <ar>                System V Release 1 archive
98091 0  string    !<arch>\n__.SYMDEF  Archive random library
98092 0  string    !<arch>             Archive
98093 0  string    ARF_BEGARF          PHIGS clear text archive
98094 0  long      0x137A2950          Scalable OpenFont binary
98095 0  long      0x137A2951          Encrypted scalable OpenFont binary
```


The use of a basic integer data type is intended to allow the implementation to choose a word size commonly used by applications on that architecture.

Earlier versions of this standard allowed for implementations with bytes other than eight bits, but this has been modified in this version.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

ar, *ls*, *pax*, *printf*

XBD Table 5-1 (on page 113), Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is marked as part of the User Portability Utilities option.

Options and an EXTENDED DESCRIPTION are added as specified in the IEEE P1003.2b draft standard.

IEEE PASC Interpretations 1003.2 #192 and #178 are applied.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/25 is applied, making major changes to address ambiguities raised in defect reports.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/26 is applied, making it clear in the OPTIONS section that the **−m**, **−d**, and **−M** options do not comply with Guideline 11 of the Utility Syntax Guidelines.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/10 is applied, clarifying the specification characters.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/11 is applied, allowing application developers to create portable magic files that can match characters in strings, and allowing common extensions found in existing implementations.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/12 is applied, removing text describing behavior on systems with bytes consisting of more than eight bits.

Issue 7

Austin Group Interpretation 1003.1-2001 #092 is applied.

SD5-XCU-ERN-4 is applied, adding further entries in the Notes column in Table 3-10.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The *file* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

The EXAMPLES section is revised to correct an error with the pathname `"$1"`.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being

used.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1141 is applied, changing ``core file'' to ``file containing a core image''.

98141 **NAME**

98142 find — find files

98143 **SYNOPSIS**

98144 `find [-H|-L] path... [operand_expression...]`

98145 **DESCRIPTION**

98146 The *find* utility shall recursively descend the directory hierarchy from each file specified by *path*,
98147 evaluating a Boolean expression composed of the primaries described in the OPERANDS section
98148 for each file encountered. Each *path* operand shall be evaluated unaltered as it was provided,
98149 including all trailing <slash> characters; all pathnames for other files encountered in the
98150 hierarchy shall consist of the concatenation of the current *path* operand, a <slash> if the current
98151 *path* operand did not end in one, and the filename relative to the *path* operand. The relative
98152 portion shall contain no dot or dot-dot components, no trailing <slash> characters, and only
98153 single <slash> characters between pathname components.

98154 The *find* utility shall be able to descend to arbitrary depths in a file hierarchy and shall not fail
98155 due to path length limitations (unless a *path* operand specified by the application exceeds
98156 {PATH_MAX} requirements).

98157 The *find* utility shall detect infinite loops; that is, entering a previously visited directory that is an
98158 ancestor of the last file encountered. When it detects an infinite loop, *find* shall write a
98159 diagnostic message to standard error and shall either recover its position in the hierarchy or
98160 terminate. In either case, the final exit status shall be non-zero.

98161 If a file is removed from or added to the directory hierarchy being searched it is unspecified
98162 whether or not *find* includes that file in its search.

98163 **OPTIONS**

98164 The *find* utility shall conform to XBD Section 12.2 (on page 215).

98165 The following options shall be supported by the implementation:

98166 **–H** Cause the file information and file type evaluated for each symbolic link
98167 encountered as a *path* operand on the command line to be those of the file
98168 referenced by the link, and not the link itself. If the referenced file does not exist,
98169 the file information and type shall be for the link itself. File information and type
98170 for symbolic links encountered during the traversal of a file hierarchy shall be that
98171 of the link itself.

98172 **–L** Cause the file information and file type evaluated for each symbolic link
98173 encountered as a *path* operand on the command line or encountered during the
98174 traversal of a file hierarchy to be those of the file referenced by the link, and not the
98175 link itself. If the referenced file does not exist, the file information and type shall be
98176 for the link itself.

98177 Specifying more than one of the mutually-exclusive options **–H** and **–L** shall not be considered
98178 an error. The last option specified shall determine the behavior of the utility. If neither the **–H**
98179 nor the **–L** option is specified, then the file information and type for symbolic links encountered
98180 as a *path* operand on the command line or encountered during the traversal of a file hierarchy
98181 shall be that of the link itself.

98182 **OPERANDS**

98183 The following operands shall be supported:

98184 The first operand and subsequent operands up to but not including the first operand that starts
98185 with a `'-'`, or is a `'!'` or a `'('`, shall be interpreted as *path* operands. If the first operand starts
98186 with a `'-'`, or is a `'!'` or a `'('`, the behavior is unspecified. Each *path* operand is a pathname of

a starting point in the file hierarchy.

The first operand that starts with a '−', or is a '!' or a '(', and all subsequent arguments shall be interpreted as an *expression* made up of the following primaries and operators. In the descriptions, wherever *n* is used as a primary argument, it shall be interpreted as a decimal integer optionally preceded by a <plus-sign> ('+') or <hyphen-minus> ('−'), as follows:

+n More than *n*.

n Exactly *n*.

−n Less than *n*.

The following primaries shall be supported:

−name *pattern*
The primary shall evaluate as true if the basename of the current pathname matches *pattern* using the pattern matching notation described in Section 2.14 (on page 2523). The additional rules in Section 2.14.3 (on page 2525) do not apply as this is a matching operation, not an expansion.

−iname *pattern*
The **−iname** primary shall be equivalent to **−name**, except that the match shall be case insensitive. See XBD Section 4.1 (on page 95).

−path *pattern*
The primary shall evaluate as true if the current pathname matches *pattern* using the pattern matching notation described in Section 2.14 (on page 2523). The additional rules in Section 2.14.3 (on page 2525) do not apply as this is a matching operation, not an expansion.

−nouser The primary shall evaluate as true if the file belongs to a user ID for which the *getpwuid*() function defined in the System Interfaces volume of POSIX.1-2024 (or equivalent) returns NULL.

−nogroup The primary shall evaluate as true if the file belongs to a group ID for which the *getgrgid*() function defined in the System Interfaces volume of POSIX.1-2024 (or equivalent) returns NULL.

−mount The primary shall always evaluate as true; it shall cause *find* to act only on files that have the same device ID (*st_dev*, see XSH *fstatat*()) as the *path* operand below which they are encountered and cause *find* not to descend below directories that have a different device ID than that *path* operand. If any **−mount** primary is specified, it shall apply to the entire expression even if the **−mount** primary would not normally be evaluated.

−xdev The primary shall always evaluate as true; it shall cause *find* not to descend below directories that have a different device ID (*st_dev*, see XSH *fstatat*()) than the *path* operand below which they are encountered; that is, when a directory with a different device ID is encountered, *find* shall act on the directory itself (unless **−mount** is specified) but shall not act on any files below the directory. If any **−xdev** primary is specified, it shall apply to the entire expression even if the **−xdev** primary would not normally be evaluated.

−prune The primary shall always evaluate as true; it shall cause *find* not to descend the current pathname if it is a directory. If the **−depth** primary is specified, the **−prune** primary shall have no effect.

98231 –**perm** **[**–**]***mode*

98232 The *mode* argument is used to represent file mode bits. It shall be processed in an
98233 identical manner to the *symbolic_mode* operand described in *chmod*, except that:

98234 1. The changes to file mode bits shall be applied to a template instead of to any
98235 files. The template shall initially have all file mode bits cleared.

98236 2. The *op* symbol `'-'` cannot be the first character of *mode*; this avoids
98237 ambiguity with the optional leading <hyphen-minus>. Since the initial
98238 mode is all bits off, there are not any symbolic modes that need to use `'-'`
98239 as the first character.

98240 If the <hyphen-minus> is omitted, the primary shall evaluate as true when the file
98241 permission bits exactly match the value of the resulting template.

98242 Otherwise, if *mode* is prefixed by a <hyphen-minus>, the primary shall evaluate as
98243 true if at least all the bits in the resulting template are set in the file permission bits.

98244 –**perm** **[**–**]***onum*

98245 If the <hyphen-minus> is omitted, the primary shall evaluate as true when the file
98246 mode bits exactly match the value of the octal number *onum* (see the description of
98247 the octal *mode* in *chmod*). Otherwise, if *onum* is prefixed by a <hyphen-minus>, the
98248 primary shall evaluate as true if at least all of the bits specified in *onum* are set. In
98249 both cases, the behavior is unspecified when *onum* exceeds 07777.

98250 –**type** *c* The primary shall evaluate as true if the type of the file is *c*, where *c* is `'b'`, `'c'`,
98251 `'d'`, `'l'`, `'p'`, `'f'`, or `'s'` for block special file, character special file, directory,
98252 symbolic link, FIFO, regular file, or socket, respectively.

98253 –**links** *n* The primary shall evaluate as true if the file has *n* links.

98254 –**user** *uname* The primary shall evaluate as true if the file belongs to the user *uname.* If *uname* is
98255 a decimal integer and the *getpwnam*() (or equivalent) function does not return a
98256 valid user name, *uname* shall be interpreted as a user ID.

98257 –**group** *gname*

98258 The primary shall evaluate as true if the file belongs to the group *gname*. If *gname*
98259 is a decimal integer and the *getgrnam*() (or equivalent) function does not return a
98260 valid group name, *gname* shall be interpreted as a group ID.

98261 –**size** *n***[c]** The primary shall evaluate as true if the file size in bytes, divided by 512 and
98262 rounded up to the next integer, is *n*. If *n* is followed by the character `'c'`, the size
98263 shall be in bytes.

98264 –**atime** *n* The primary shall evaluate as true if the file access time subtracted from the
98265 initialization time, divided by 86 400 (with any remainder discarded), is *n*.

98266 –**ctime** *n* The primary shall evaluate as true if the time of last change of file status
98267 information subtracted from the initialization time, divided by 86 400 (with any
98268 remainder discarded), is *n*.

98269 –**mtime** *n* The primary shall evaluate as true if the file modification time subtracted from the
98270 initialization time, divided by 86 400 (with any remainder discarded), is *n*.

98271 –**exec** *utility_name* **[***argument* ...**] ;**
98272 –**exec** *utility_name* **[***argument* ...**]** {} +

98273 The end of the primary expression shall be punctuated by a <semicolon> or by a
98274 <plus-sign>. Only a <plus-sign> that immediately follows an argument
98275 containing only the two characters `"{}"` shall punctuate the end of the primary

98276 expression. Other uses of the <plus-sign> shall not be treated as special.

98277 If the primary expression is punctuated by a <semicolon>, the utility *utility_name*
98278 shall be invoked once for each pathname and the primary shall evaluate as true if
98279 the utility returns a zero value as exit status. A *utility_name* or *argument* containing
98280 only the two characters `"{}"` shall be replaced by the current pathname. If a
98281 *utility_name* or *argument* string contains the two characters `"{}"`, but not just the
98282 two characters `"{}"`, it is implementation-defined whether *find* replaces those two
98283 characters or uses the string without change.

98284 If the primary expression is punctuated by a <plus-sign>, the primary shall always
98285 evaluate as true, and the pathnames for which the primary is evaluated shall be
98286 aggregated into sets. The utility *utility_name* shall be invoked once for each set of
98287 aggregated pathnames. Each invocation shall begin after the last pathname in the
98288 set is aggregated, and shall be completed before the *find* utility exits and before the
98289 first pathname in the next set (if any) is aggregated for this primary, but it is
98290 otherwise unspecified whether the invocation occurs before, during, or after the
98291 evaluations of other primaries. If any invocation returns a non-zero value as exit
98292 status, the *find* utility shall return a non-zero exit status. An argument containing
98293 only the two characters `"{}"` shall be replaced by the set of aggregated
98294 pathnames, with each pathname passed as a separate argument to the invoked
98295 utility in the same order that it was aggregated. The size of any set of two or more
98296 pathnames shall be limited such that execution of the utility does not cause the
98297 system's {ARG_MAX} limit to be exceeded. If more than one argument containing
98298 the two characters `"{}"` is present, the behavior is unspecified.

98299 The current directory for the invocation of *utility_name* shall be the same as the
98300 current directory when the *find* utility was started. If the *utility_name* names any of
98301 the special built-in utilities (see Section 2.15, on page 2526), the results are
98302 undefined.

98303 −**ok** *utility_name* [*argument* ...] **;**

98304 The −**ok** primary shall be equivalent to −**exec**, except that the use of a <plus-sign>
98305 to punctuate the end of the primary expression need not be supported, and *find*
98306 shall request affirmation of the invocation of *utility_name* using the current file as
98307 an argument by writing to standard error as described in the STDERR section. If
98308 the response on standard input is affirmative, the utility shall be invoked.
98309 Otherwise, the command shall not be invoked and the value of the −**ok** operand
98310 shall be false.

98311 −**print** The primary shall always evaluate as true; it shall cause the current pathname to
98312 be written to standard output, followed by a <newline>.

98313 −**print0** The primary shall always evaluate as true; it shall cause the current pathname to
98314 be written to standard output, followed by a null byte.

98315 −**newer** *file* The primary shall evaluate as true if the modification time of the current file is
98316 more recent than the modification time of the file named by the pathname *file*. If
98317 *file* names a symbolic link, the modification time used shall be that of the file
98318 referenced by the symbolic link if either the −**H** or −**L** option is specified; if neither
98319 −**H** nor −**L** is specified, it is unspecified whether the modification time is that of the
98320 symbolic link itself or of the file referenced by the symbolic link. In either case, if
98321 the referenced file does not exist, the modification time used shall be that of the
98322 link itself. If *file* is a relative pathname, it shall be resolved relative to the current
98323 working directory that was inherited by *find* when it was invoked.

98324 –**depth** The primary shall always evaluate as true; it shall cause descent of the directory
98325 hierarchy to be done so that all entries in a directory are acted on before the
98326 directory itself. If a –**depth** primary is not specified, all entries in a directory shall
98327 be acted on after the directory itself. If any –**depth** primary is specified, it shall
98328 apply to the entire expression even if the –**depth** primary would not normally be
98329 evaluated.

98330 The primaries can be combined using the following operators (in order of decreasing
98331 precedence):

98332 (*expression*) True if *expression* is true.

98333 **!** *expression* Negation of a primary; the unary NOT operator.

98334 *expression* **[**–**a]** *expression*
98335 Conjunction of primaries; the AND operator is implied by the juxtaposition of two
98336 primaries or made explicit by the optional –**a** operator. The second expression shall
98337 not be evaluated if the first expression is false.

98338 *expression* –**o** *expression*
98339 Alternation of primaries; the OR operator. The second expression shall not be
98340 evaluated if the first expression is true.

98341 If no *expression* is present, –**print** shall be used as the expression. Otherwise, if the given
98342 expression does not contain any of the primaries –**exec**, –**ok**, or –**print**, the given expression
98343 shall be effectively replaced by:

98344 `( given_expression ) -print`

98345 The –**user**, –**group**, and –**newer** primaries each shall evaluate their respective arguments only
98346 once.

98347 When the file type evaluated for the current file is a symbolic link, the results of evaluating the
98348 –**perm** primary are implementation-defined.

98349 **STDIN**

98350 If the –**ok** primary is used, the response shall be read from the standard input. An entire line
98351 shall be read as the response. Otherwise, the standard input shall not be used.

98352 **INPUT FILES**

98353 None.

98354 **ENVIRONMENT VARIABLES**

98355 The following environment variables shall affect the execution of *find*:

98356 *LANG* Provide a default value for the internationalization variables that are unset or null.
98357 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
98358 variables used to determine the values of locale categories.)

98359 *LC_ALL* If set to a non-empty string value, override the values of all the other
98360 internationalization variables.

98361 *LC_COLLATE*
98362 Determine the locale for the behavior of ranges, equivalence classes, and multi-
98363 character collating elements used in the pattern matching notation for the –**name**,
98364 –**iname**, and –**path** primaries and in the extended regular expression defined for
98365 the **yesexpr** locale keyword in the *LC_MESSAGES* category.

LC_CTYPE This variable determines the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments), the behavior of character classes within the pattern matching notation used for the −**name**, −**iname**, and −**path** primaries, and the behavior of character classes within regular expressions used in the extended regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES* category.

LC_MESSAGES
Determine the locale used to process affirmative responses, and the locale used to affect the format and contents of diagnostic messages and prompts written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

PATH Determine the location of the *utility_name* for the −**exec** and −**ok** primaries, as described in XBD Chapter 8 (on page 167).

ASYNCHRONOUS EVENTS

Default.

STDOUT

The −**print** primary shall cause the current pathname to be written to standard output. The format shall be:

```
"%s\n", <path>
```

The −**print0** primary shall cause the current pathname to be written to standard output, followed by a null byte.

STDERR

The −**ok** primary shall write a prompt to standard error containing at least the *utility_name* to be invoked and the current pathname. In the POSIX locale, the last non-<blank> in the prompt shall be `'?'`. The exact format used is unspecified.

Otherwise, the standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All *path* operands were traversed successfully, the output (if any) specified in STDOUT was successfully written to standard output, and all commands (if any) executed using the −**exec** primary punctuated by a <plus-sign> exited with exit status 0.

>0 A command executed using the −**exec** primary punctuated by a <plus-sign> exited with non-zero status, or an error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

When used in operands, pattern matching notation, <semicolon>, <left-parenthesis>, and <right-parenthesis> characters are special to the shell and must be quoted (see Section 2.2, on page 2472).

When restricting the search to files on one file system, it can sometimes be desirable for the crossing points themselves to be acted on and sometimes for them not to be acted on. (Crossing points are mount points and, if the **–L** option is specified, symbolic links to directories on other file systems.) The **–xdev** primary acts on them and the **–mount** primary does not. However, **–mount** also does not act on symbolic links to non-directory files on other file systems (if **–L** is specified). If there is a need for an application to exclude crossing points but include symbolic links to non-directory files on other file systems, this can be achieved by using two *find* commands as follows:

```
find –L dir –mount –type d –print
find –L dir –xdev ! –type d –print
```

(in a subshell whose output is piped to *sort*, if the order matters).

If both **–mount** and **–xdev** are specified, *find* obeys both primaries but the end result is the same as if **–xdev** were not specified.

The bit that is traditionally used for sticky (historically 01000) is specified in the **–perm** primary using the octal number argument form. Since this bit is not defined by this volume of POSIX.1-2024, applications must not assume that it actually refers to the traditional sticky bit.

EXAMPLES

1. The following commands are equivalent:

   ```
   find .
   find . –print
   ```

   They both write out the entire directory hierarchy from the current directory.

   With this output format, if any pathnames include <newline> characters, it is not possible to tell where each pathname begins and ends. This problem can be avoided by omitting such pathnames:

   ```
   LC_ALL=POSIX find . –name $'*\n*' –prune –o –print
   ```

   or by using a sentinel in the pathname that *find* would never otherwise produce, such as:

   ```
   find .//. –print
   ```

   or by using **–print0** instead of **–print** and processing the output with a utility that can accept null-terminated pathnames as input, such as *xargs* with the **–0** option or *read* with **–d ""**, for example:

   ```
   find . –print0 | while IFS= read –rd "" file
   do
       # process "$file"
   done
   ```

   It should be noted that using *find* with **–print0** to pipe input to *xargs* **–r0** is less safe than using *find* with **–exec** because if *find* **–print0** is terminated after it has written a partial pathname, the partial pathname may be processed as if it was a complete pathname.

2. The following command:

```
find / \( -name tmp -o -name '*.xx' \) ! -type d -mtime +7 \
    -exec rm {} +
```

removes all files named **tmp** or ending in **.xx** that have not been modified for more than seven (that is, eight or more) 24-hour periods.

3. The following command:

```
find . -perm -o+w,+s
```

prints (–**print** is assumed) the names of all files in or below the current directory, with all of the file permission bits S_ISUID, S_ISGID, and S_IWOTH set, regardless of the value of the file creation mask. (Note that the file creation mask is only specified for the file permission bits, and not S_ISUID, S_ISGID or S_ISVTX.)

4. The following command:

```
find . -perm -+w
```

prints (–**print** is assumed) the names of all files in or below the current directory, with S_IWUSR set if the file creation mask does not have S_IWUSR set (otherwise the S_IWUSR bit is ignored), S_IWGRP set if the file creation mask does not have S_IWGRP set (otherwise S_IWGRP is ignored), and S_IWOTH set if the file creation mask does not have S_IWOTH set (otherwise S_IWOTH is ignored).

5. The following command:

```
find . -name SCCS -prune -o -print
```

recursively prints pathnames of all files in the current directory and below, but skips directories named SCCS and files in them.

6. The following command:

```
find . -print -name SCCS -prune
```

behaves as in the previous example, but prints the names of the SCCS directories.

7. The following command is roughly equivalent to the –**nt** extension to *test*:

```
if [ -n "$(find file1 -prune -newer file2)" ]; then
    printf %s\\n "file1 is newer than file2"
fi
```

8. The descriptions of –**atime**, –**ctime**, and –**mtime** use the terminology *n* ``86 400 second periods (days)''. For example, a file accessed at 23:59 is selected by:

```
find . -atime -1 -print
```

at 00:01 the next day (less than 24 hours later, not more than one day ago); the midnight boundary between days has no effect on the 24-hour calculation.

9. The following command:

```
find . ! -name . -prune -name '*.old' -exec \
    sh -c 'mv "$@" ../old/' sh {} +
```

performs the same task as:

```
mv ./*.old ./.old ./.*.old ../old/
```

while avoiding an ``Argument list too long'' error if there are a large number of files

ending with **.old** and without running *mv* if there are no such files (and avoiding ``No such file or directory'' errors if **./.old** does not exist or no files match **./*.old** or **./.*.old**).

The alternative:

```
find . ! -name . -prune -name '*.old' -exec mv {} ../old/ \;
```

is less efficient if there are many files to move because it executes one *mv* command per file.

10. On systems configured to mount removable media on directories under **/media**, the following command searches the file hierarchy for files of size larger than 100 000 KiB without searching any mounted removable media:

```
find / -path /media -prune -o -size +200000 -print
```

11. Except for the root directory, and `"//"` on implementations where `"//"` does not refer to the root directory, no pattern given to **-name** will match a <slash>, because trailing <slash> characters are ignored when computing the basename of the file under evaluation. Given two empty directories named **foo** and **bar**, the following command:

```
find foo/// bar/// -name foo -o -name 'bar?*'
```

prints only the line `"foo///"`.

RATIONALE

The **-a** operator was retained as an optional operator for compatibility with historical shell scripts, even though it is redundant with expression concatenation.

The descriptions of the `'-'` modifier on the *mode* and *onum* arguments to the **-perm** primary agree with historical practice on BSD and System V implementations. System V and BSD documentation both describe it in terms of checking additional bits; in fact, it uses the same bits, but checks for having at least all of the matching bits set instead of having exactly the matching bits set.

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified because:

- Implementations may desire more descriptive prompts than those used on historical implementations.
- Since the historical prompt strings do not terminate with <newline> characters, there is no portable way for another program to interact with the prompts of this utility via pipes.

Therefore, an application using this prompting option relies on the system to provide the most suitable dialog directly with the user, based on the general guidelines specified.

The **-size** operand refers to the size of a file, rather than the number of blocks it may occupy in the file system. The intent is that the *st_size* field defined in the System Interfaces volume of POSIX.1-2024 should be used, not the *st_blocks* found in historical implementations. There are at least two reasons for this:

1. In both System V and BSD, *find* only uses *st_size* in size calculations for the operands specified by this volume of POSIX.1-2024. (BSD uses *st_blocks* only when processing the **-ls** primary.)
2. Users usually think of file size in terms of bytes, which is also the unit used by the *ls* utility for the output from the **-l** option. (In both System V and BSD, *ls* uses *st_size* for the **-l** option size field and uses *st_blocks* for the *ls* **-s** calculations. This volume of POSIX.1-2024 does not specify *ls* **-s**.)

The descriptions of –**atime**, –**ctime**, and –**mtime** were changed from the SVID description of *n* ``days'' to *n* being the result of the integer division of the time difference in seconds by 86 400. The description is also different in terms of the exact timeframe for the *n* case (*versus* the +*n* or –*n*), but it matches all known historical implementations. It refers to one 86 400 second period in the past, not any time from the beginning of that period to the current time. For example, –**atime** 2 is true if the file was accessed any time in the period from 72 hours to 48 hours ago.

Historical implementations do not modify `"{}"` when it appears as a substring of an –**exec** or –**ok** *utility_name* or argument string. There have been numerous user requests for this extension, so this volume of POSIX.1-2024 allows the desired behavior. At least one recent implementation does support this feature, but encountered several problems in managing memory allocation and dealing with multiple occurrences of `"{}"` in a string while it was being developed, so it is not yet required behavior.

Assuming the presence of –**print** was added to correct a historical pitfall that plagues novice users, it is entirely upwards-compatible from the historical System V *find* utility. In its simplest form (*find directory*), it could be confused with the historical BSD fast *find*. The BSD developers agreed that adding –**print** as a default expression was the correct decision and have added the fast *find* functionality within a new utility called *locate*.

Historically, the –**L** option was implemented using the primary –**follow**. The –**H** and –**L** options were added for two reasons. First, they offer a finer granularity of control and consistency with other programs that walk file hierarchies. Second, the –**follow** primary always evaluated to true. As they were historically really global variables that took effect before the traversal began, some valid expressions had unexpected results. An example is the expression –**print** –**o** –**follow**. Because –**print** always evaluates to true, the standard order of evaluation implies that –**follow** would never be evaluated. This was never the case. Historical practice for the –**follow** primary, however, is not consistent. Some implementations always follow symbolic links on the command line whether –**follow** is specified or not. Others follow symbolic links on the command line only if –**follow** is specified. Both behaviors are provided by the –**H** and –**L** options, but scripts using the current –**follow** primary would be broken if the –**follow** option is specified to work either way.

Since the –**L** option resolves all symbolic links and the –**type** *l* primary is true for symbolic links that still exist after symbolic links have been resolved, the command:

```
find -L . -type l
```

prints a list of symbolic links reachable from the current directory that do not resolve to accessible files.

A feature of SVR4's *find* utility was the –**exec** primary's **+** terminator. This allowed filenames containing special characters (especially <newline> characters) to be grouped together without the problems that occur if such filenames are piped to *xargs*.

The `"-exec ... {} +"` syntax adopted was a result of IEEE PASC Interpretation 1003.2 #210. It should be noted that this is an incompatible change to IEEE Std 1003.2-1992. For example, the following command printed all files with a `'-'` after their name if they are regular files, and a `'+'` otherwise:

```
find / -type f -exec echo {} - ';' -o -exec echo {} + ';'
```

The change invalidates usage like this. Even though the previous standard stated that this usage would work, in practice many did not support it and the standard developers felt it better to now state that this was not allowable.

Historically, many *find* implementations supported –**mount** and –**xdev** as synonymous

primaries and earlier versions of this standard only required support for –**xdev**. However, the behavior of *find* with –**xdev** differed from that of the *nftw*( ) function with FTW_MOUNT as regards whether the mount point itself was included or excluded. Therefore the standard now requires support for both primaries with slightly differing behaviors: –**mount** behaves in the manner of *nftw*( ) with the traditional FTW_MOUNT flag, and –**xdev** in the manner of *nftw*( ) with a new FTW_XDEV flag.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

Section 2.2 (on page 2472), Section 2.14 (on page 2523), Section 2.15 (on page 2526), *chmod*, *mv*, *pax*, *sh*, *test*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *fstatat*( ), *getgrgid*( ), *getpwuid*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The –**perm** [–]*onum* primary is supported.

The *find* utility is aligned with the IEEE P1003.2b draft standard, to include processing of symbolic links and changes to the description of the **atime**, **ctime**, and **mtime** operands.

IEEE PASC Interpretation 1003.2 #210 is applied, extending the –**exec** operand.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/13 is applied, updating the RATIONALE section to be consistent with the normative text.

Issue 7

Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the *LC_MESSAGES* environment variable.

Austin Group Interpretation 1003.1-2001 #127 is applied, rephrasing the description of the –**exec** primary to be ``immediately follows''.

Austin Group Interpretation 1003.1-2001 #185 is applied, clarifying the requirements for the –**H** and –**L** options.

Austin Group Interpretation 1003.1-2001 #186 is applied, clarifying the requirements for the evaluation of *path* operands.

Austin Group Interpretation 1003.1-2001 #195 is applied, clarifying the interpretation of the first operand.

SD5-XCU-ERN-48 is applied, clarifying the –**L** option in the case that the referenced file does not exist.

SD5-XCU-ERN-89 is applied, updating the OPERANDS section.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

SD5-XCU-ERN-117 is applied, clarifying the –**perm** operand.

SD5-XCU-ERN-122 is applied, adding a new EXAMPLE.

The description of the –**name** primary is revised and the –**path** primary is added (with a new example).

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0086 [365], XCU/TC1-2008/0087 [310], XCU/TC1-2008/0088 [309,310,430], XCU/TC1-2008/0089 [235], and XCU/TC1-2008/0090 [445] are applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0099 [584], XCU/TC2-2008/0100 [584], and XCU/TC2-2008/0101 [584] are applied.

Issue 8

Austin Group Defect 243 is applied, adding the –**print0** primary.

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1031 is applied, adding the –**iname** primary.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1133 is applied, adding the –**mount** primary.

Austin Group Defects 1259 and 1777 are applied, changing the EXAMPLES section.

Austin Group Defect 1392 is applied, changing the effect of the file creation mask on the *mode* argument for the –**perm** primary to be consistent with *chmod*.

Austin Group Defect 1501 is applied, changing the EXIT STATUS section.

Austin Group Defect 1553 is applied, changing the ENVIRONMENT VARIABLES section.

Austin Group Defect 1554 is applied, changing the RATIONALE section.

Austin Group Defect 1606 is applied, clarifying that if *find* detects an infinite loop and recovers its position, the final exit status is non-zero.

Austin Group Defect 1776 is applied, clarifying how symbolic links are handled by the –**newer** *file* primary.

98648 **NAME**

98649 fold — filter for folding lines

98650 **SYNOPSIS**

98651 `fold [-bs] [-w width] [file...]`

98652 **DESCRIPTION**

98653 The *fold* utility is a filter that shall fold lines from its input files, breaking the lines to have a
98654 maximum of *width* column positions (or bytes, if the **–b** option is specified). Lines shall be
98655 broken by the insertion of a <newline> such that each output line (referred to later in this section
98656 as a *segment*) is the maximum width possible that does not exceed the specified number of
98657 column positions (or bytes). A line shall not be broken in the middle of a character. The behavior
98658 is undefined if *width* is less than the number of columns any single character in the input would
98659 occupy.

98660 If the <carriage-return>, <backspace>, or <tab> characters are encountered in the input, and the
98661 **–b** option is not specified, they shall be treated specially:

98662 <backspace> The current count of line width shall be decremented by one, although the count
98663 never shall become negative. The *fold* utility shall not insert a <newline>
98664 immediately before or after any <backspace>, unless the following character has a
98665 width greater than 1 and would cause the line width to exceed *width*.

98666 <carriage-return>
98667 The current count of line width shall be set to zero. The *fold* utility shall not insert a
98668 <newline> immediately before or after any <carriage-return>.

98669 <tab> Each <tab> encountered shall advance the column position pointer to the next tab
98670 stop. Tab stops shall be at each column position *n* such that *n* modulo 8 equals 1.

98671 **OPTIONS**

98672 The *fold* utility shall conform to XBD Section 12.2 (on page 215).

98673 The following options shall be supported:

98674 **–b** Count *width* in bytes rather than column positions.

98675 **–s** If a segment of a line contains a <blank> within the first *width* column positions (or
98676 bytes), break the line after the last such <blank> meeting the width constraints. If
98677 there is no <blank> meeting the requirements, the **–s** option shall have no effect for
98678 that output segment of the input line.

98679 **–w** *width* Specify the maximum line length, in column positions (or bytes if **–b** is specified).
98680 The results are unspecified if *width* is not a positive decimal number. The default
98681 value shall be 80.

98682 **OPERANDS**

98683 The following operand shall be supported:

98684 *file* A pathname of a text file to be folded. If no *file* operands are specified, the standard
98685 input shall be used.

98686 **STDIN**

98687 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*
98688 operand is `'-'` and the implementation treats the `'-'` as meaning standard input. Otherwise,
98689 the standard input shall not be used. See the INPUT FILES section.

98690 **INPUT FILES**

98691 If the **–b** option is specified, the input files shall be text files except that the lines are not limited
98692 to {LINE_MAX} bytes in length. If the **–b** option is not specified, the input files shall be text files.

98693 **ENVIRONMENT VARIABLES**

98694 The following environment variables shall affect the execution of *fold*:

98695 *LANG* Provide a default value for the internationalization variables that are unset or null.
98696 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
98697 variables used to determine the values of locale categories.)

98698 *LC_ALL* If set to a non-empty string value, override the values of all the other
98699 internationalization variables.

98700 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
98701 characters (for example, single-byte as opposed to multi-byte characters in
98702 arguments and input files), and for the determination of the width in column
98703 positions each character would occupy on a constant-width font output device.

98704 *LC_MESSAGES*
98705 Determine the locale that should be used to affect the format and contents of
98706 diagnostic messages written to standard error.

98707 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

98708 **ASYNCHRONOUS EVENTS**

98709 Default.

98710 **STDOUT**

98711 The standard output shall be a file containing a sequence of characters whose order shall be
98712 preserved from the input files, possibly with inserted <newline> characters.

98713 **STDERR**

98714 The standard error shall be used only for diagnostic messages.

98715 **OUTPUT FILES**

98716 None.

98717 **EXTENDED DESCRIPTION**

98718 None.

98719 **EXIT STATUS**

98720 The following exit values shall be returned:

98721 0 All input files were processed successfully.

98722 >0 An error occurred.

98723 **CONSEQUENCES OF ERRORS**

98724 Default.

APPLICATION USAGE

The *cut* and *fold* utilities can be used to create text files out of files with arbitrary line lengths. The *cut* utility should be used when the number of lines (or records) needs to remain constant. The *fold* utility should be used when the contents of long lines need to be kept contiguous.

The *fold* utility is frequently used to send text files to printers that truncate, rather than fold, lines wider than the printer is able to print (usually 80 or 132 column positions).

EXAMPLES

An example invocation that submits a file of possibly long lines to the printer (under the assumption that the user knows the line width of the printer to be assigned by *lp*):

```
fold -w 132 bigfile | lp
```

RATIONALE

Although terminal input in canonical processing mode requires the erase character (frequently set to <backspace>) to erase the previous character (not byte or column position), terminal output is not buffered and is extremely difficult, if not impossible, to parse correctly; the interpretation depends entirely on the physical device that actually displays/prints/stores the output. In all known internationalized implementations, the utilities producing output for mixed column-width output assume that a <backspace> character backs up one column position and outputs enough <backspace> characters to return to the start of the character when <backspace> is used to provide local line motions to support underlining and emboldening operations. Since *fold* without the **–b** option is dealing with these same constraints, <backspace> is always treated as backing up one column position rather than backing up one character.

Historical versions of the *fold* utility assumed 1 byte was one character and occupied one column position when written out. This is no longer always true. Since the most common usage of *fold* is believed to be folding long lines for output to limited-length output devices, this capability was preserved as the default case. The **–b** option was added so that applications could *fold* files with arbitrary length lines into text files that could then be processed by the standard utilities. Note that although the width for the **–b** option is in bytes, a line is never split in the middle of a character. (It is unspecified what happens if a width is specified that is too small to hold a single character found in the input followed by a <newline>.)

The tab stops are hardcoded to be every eighth column to meet historical practice. No new method of specifying other tab stops was invented.

FUTURE DIRECTIONS

None.

SEE ALSO

cut

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 6

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #092 is applied.

Austin Group Interpretation 1003.1-2001 #204 is applied, updating the DESCRIPTION to clarify when a <newline> can be inserted before or after a <backspace>.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

98772 **NAME**

98773 fuser — list process IDs of all processes that are using one or more named files

98774 **SYNOPSIS**

98775 XSI
```
fuser [-cfu] file...
```

98776 **DESCRIPTION**

98777 For each *file* operand, in order, *fuser* shall write one line of output, some of it to standard output,
98778 and the rest to standard error, giving information about processes running on the local system
98779 that are using the file. A process shall be considered to be using a file if it has at least one open
98780 file descriptor associated with the file or if the file is a directory that is the current working
98781 directory or the root directory for the process, and may be considered to be using a file for other
98782 implementation-dependent reasons. If *file* names a block special device that contains a mounted
98783 file system, and the –**f** option is not specified, any processes using any file on that mounted file
98784 system and any processes that are using the device file itself shall be listed.

98785 Any output for processes running on remote systems that are using a named file is unspecified.

98786 A user may need appropriate privileges to invoke the *fuser* utility.

98787 When standard output and standard error are directed to the same file, the output for each *file*
98788 operand shall be interleaved so that it is written to the file in the following order:

98789 • On standard error, a pathname for the file, immediately followed by a <colon> and zero or
98790 more <blank> characters. The pathname shall be either the file operand (unaltered) or the
98791 pathname that would result from a successful call to the *realpath*( ) function, defined in the
98792 System Interfaces volume of POSIX.1-2024, with the *file* operand as its *file_name* argument.

98793 • For each process using the file:

98794 — On standard output, the process ID in the format:

98795 `" %1d", <process ID>`

98796 — On standard error, information about the file's use by the process, in the following
98797 format:

98798 `"%s", <use chars>`

98799 if the –**u** option is not specified, or in the following format:

98800 `"%s(%s)", <use chars>, <user name>`

98801 if the –**u** option is specified, where <*use chars*> is a string of zero or more characters
98802 indicating the use of the file and <*user name*> is the user name corresponding to the
98803 real user ID of the process or, if the user name cannot be resolved from the real user
98804 ID of the process, the real user ID of the process in decimal. The value of <*use chars*>
98805 shall include the character `'c'` if the process is using the file as its current directory
98806 and the character `'r'` if the process is using the file as its root directory;
98807 implementations may include other alphabetic characters to indicate other uses of
98808 the file.

98809 • On standard error, a <newline> character.

98810 When standard output and standard error are not directed to the same file, the data written to
98811 each shall be as described above but the ordering of writes to standard output relative to writes
98812 to standard error is unspecified. For example, *fuser* might first write the information for all file
98813 operands to standard error and then write all of the process IDs to standard output.

98814 **OPTIONS**

98815 The *fuser* utility shall conform to XBD Section 12.2 (on page 215).

98816 The following options shall be supported:

98817 **−c** If a *file* operand names a directory that is the mount point of a mounted file system,
98818 all processes using any file on that file system shall be listed as if they were using
98819 the named directory. The behavior for any *file* operand that names an existing file
98820 that is not the mount point of a mounted file system is unspecified.

98821 **−f** The report shall be only for the named files.

98822 **−u** The user name, in parentheses, associated with each process ID written to standard
98823 output shall be written to standard error.

98824 **OPERANDS**

98825 The following operand shall be supported:

98826 *file* A pathname of a file for which the processes using the file are to be reported.

98827 **STDIN**

98828 Not used.

98829 **INPUT FILES**

98830 The user database.

98831 **ENVIRONMENT VARIABLES**

98832 The following environment variables shall affect the execution of *fuser*:

98833 *LANG* Provide a default value for the internationalization variables that are unset or null.
98834 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
98835 variables used to determine the values of locale categories.)

98836 *LC_ALL* If set to a non-empty string value, override the values of all the other
98837 internationalization variables.

98838 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
98839 characters (for example, single-byte as opposed to multi-byte characters in
98840 arguments).

98841 *LC_MESSAGES*
98842 Determine the locale that should be used to affect the format and contents of
98843 diagnostic messages written to standard error.

98844 *NLSPATH* Determine the location of messages objects and message catalogs.

98845 **ASYNCHRONOUS EVENTS**

98846 Default.

98847 **STDOUT**

98848 See DESCRIPTION.

98849 **STDERR**

98850 The *fuser* utility shall write diagnostic messages to standard error.

98851 The *fuser* utility also shall write information to standard error as specified in the DESCRIPTION
98852 section.

98853 **OUTPUT FILES**

98854 None.

98855 **EXTENDED DESCRIPTION**

98856 None.

98857 **EXIT STATUS**

98858 The following exit values shall be returned:

98859 0 Successful completion.

98860 >0 An error occurred.

98861 **CONSEQUENCES OF ERRORS**

98862 Default.

98863 **APPLICATION USAGE**

98864 Things can change while *fuser* is running; the snapshot it gives is only true for an instant, and
98865 might not be accurate by the time it is displayed.

98866 **EXAMPLES**

98867 The command:

98868 `fuser -fu .`

98869 writes to standard output the process IDs of processes that are using the current directory and
98870 writes to standard error an indication of how those processes are using the directory and the
98871 user names associated with the processes that are using the current directory.

98872 `fuser -c <mount point>`

98873 writes to standard output the process IDs of processes that are using any file in the file system
98874 which is mounted on <*mount point*> and writes to standard error an indication of how those
98875 processes are using the files.

98876 `fuser <mount point>`

98877 writes to standard output the process IDs of processes that are using the file which is named by
98878 <*mount point*> and writes to standard error an indication of how those processes are using the
98879 file.

98880 `fuser <mounted block device>`

98881 writes to standard output the process IDs of processes that are using any file on the mounted file
98882 system contained by <*mounted block device*> and of processes that are using the device file
98883 <*mounted block device*> itself, and writes to standard error an indication of how those processes
98884 are using the files.

98885 `fuser -f <mounted block device>`

98886 writes to standard output the process IDs of processes that are using the file <*mounted block*
98887 *device*> itself and writes to standard error an indication of how those processes are using the file.

98888 **RATIONALE**

98889 The definition of the *fuser* utility follows existing practice.

98890 **FUTURE DIRECTIONS**

98891 If this utility is directed to display a pathname that contains any bytes that have the encoded
98892 value of a <newline> character when <newline> is a terminator or separator in the output
98893 format being used, implementations are encouraged to treat this as an error. A future version of
98894 this standard may require implementations to treat this as an error.

SEE ALSO

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 5.

Issue 7

SD5-XCU-ERN-90 is applied, updating the EXAMPLES section.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1746 is applied, clarifying the output written to standard output and standard error.

98910 **NAME**

98911 gencat — generate a formatted message catalog

98912 **SYNOPSIS**

98913 `gencat` *catfile msgfile...*

98914 **DESCRIPTION**

98915 The *gencat* utility shall merge the message text source file *msgfile* into a formatted message
98916 catalog *catfile*. The file *catfile* shall be created if it does not already exist. If *catfile* does exist, its
98917 messages shall be included in the new *catfile*. If set and message numbers collide, the new
98918 message text defined in *msgfile* shall replace the old message text currently contained in *catfile*.

98919 **OPTIONS**

98920 None.

98921 **OPERANDS**

98922 The following operands shall be supported:

98923 *catfile* A pathname of the formatted message catalog. If `'−'` is specified, standard output
98924 shall be used. The format of the message catalog produced is unspecified.

98925 *msgfile* A pathname of a message text source file. If `'−'` is specified for an instance of
98926 *msgfile*, standard input shall be used. The format of message text source files is
98927 defined in the EXTENDED DESCRIPTION section.

98928 **STDIN**

98929 The standard input shall not be used unless a *msgfile* operand is specified as `'−'`.

98930 **INPUT FILES**

98931 The input files shall be text files.

98932 **ENVIRONMENT VARIABLES**

98933 The following environment variables shall affect the execution of *gencat*:

98934 *LANG* Provide a default value for the internationalization variables that are unset or null.
98935 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
98936 variables used to determine the values of locale categories.)

98937 *LC_ALL* If set to a non-empty string value, override the values of all the other
98938 internationalization variables.

98939 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
98940 characters (for example, single-byte as opposed to multi-byte characters in
98941 arguments and input files).

98942 *LC_MESSAGES*
98943 Determine the locale that should be used to affect the format and contents of
98944 diagnostic messages written to standard error.

98945 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

98946 **ASYNCHRONOUS EVENTS**

98947 Default.

98948 **STDOUT**

98949 The standard output shall not be used unless the *catfile* operand is specified as `'−'`.

98950 **STDERR**

98951 The standard error shall be used only for diagnostic messages.

98952 **OUTPUT FILES**

98953 None.

98954 **EXTENDED DESCRIPTION**

98955 The content of a message text file shall be in the format defined as follows. Note that the fields of
98956 a message text source line are separated by a single <blank> character. Any other <blank>
98957 characters are considered to be part of the subsequent field.

98958 **$set** *n comment*

98959 This line specifies the set identifier of the following messages until the next **$set** or
98960 end-of-file appears. The *n* denotes the set identifier, which is defined as a number
98961 in the range [1, {NL_SETMAX}] (see the **<limits.h>** header defined in the Base
98962 Definitions volume of POSIX.1-2024). The application shall ensure that set
98963 identifiers are presented in ascending order within a single source file, but need
98964 not be contiguous. Any string following the set identifier shall be treated as a
98965 comment. If no **$set** directive is specified in a message text source file, all messages
98966 shall be located in an implementation-defined default message set NL_SETD (see
98967 the **<nl_types.h>** header defined in the Base Definitions volume of POSIX.1-2024).

98968 **$delset** *n comment*

98969 This line deletes message set *n* from an existing message catalog. The *n* denotes the
98970 set number [1, {NL_SETMAX}]. Any string following the set number shall be
98971 treated as a comment.

98972 **$** *comment* A line beginning with `'$'` followed by a <blank> shall be treated as a comment.

98973 *m message-text*

98974 The *m* denotes the message identifier, which is defined as a number in the range [1,
98975 {NL_MSGMAX}] (see the **<limits.h>** header). The *message-text* shall be stored in the
98976 message catalog with the set identifier specified by the last **$set** directive, and with
98977 message identifier *m*. If the *message-text* is empty, and a <blank> field separator is
98978 present, an empty string shall be stored in the message catalog. If a message source
98979 line has a message number, but neither a field separator nor *message-text*, the
98980 existing message with that number (if any) shall be deleted from the catalog. The
98981 application shall ensure that message identifiers are in ascending order within a
98982 single set, but need not be contiguous. The application shall ensure that the length
98983 of *message-text* is in the range [0, {NL_TEXTMAX}] (see the **<limits.h>** header).

98984 **$quote** *c* This line specifies an optional quote character *c*, which can be used to surround
98985 *message-text* so that trailing <space> characters or null (empty) messages are visible
98986 in a message source line. By default, or if an empty **$quote** directive is supplied, no
98987 quoting of *message-text* shall be recognized.

98988 Empty lines in a message text source file shall be ignored. The effects of lines starting with any
98989 character other than those defined above are implementation-defined.

98990 Text strings can contain the special characters and escape sequences defined in the following
98991 table:

Description	Symbol	Sequence
<newline>	NL(LF)	\n
Horizontal-tab	HT	\t
<vertical-tab>	VT	\v
<backspace>	BS	\b
<carriage-return>	CR	\r
<form-feed>	FF	\f
Backslash	\	\\
Bit pattern	ddd	\ddd

The escape sequence "\ddd" consists of <backslash> followed by one, two, or three octal digits, which shall be taken to specify the value of the desired character. If the character following a <backslash> is not one of those specified, the <backslash> shall be ignored.

A <backslash> followed by a <newline> is also used to continue a string on the following line. Thus, the following two lines describe a single message string:

```
1 This line continues \
to the next line
```

which shall be equivalent to:

```
1 This line continues to the next line
```

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Message catalogs produced by *gencat* are binary encoded, meaning that their portability cannot be guaranteed between different types of machine. Thus, just as C programs need to be recompiled for each type of machine, so message catalogs must be recreated via *gencat*.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iconv

XBD Chapter 8 (on page 167), **<limits.h>**, **<nl_types.h>**

CHANGE HISTORY

First released in Issue 3.

Issue 6

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

The *gencat* utility is moved from the XSI option to the Base.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1463 is applied, changing *n* to *c* in the definition of **$quote**.

Austin Group Defect 1516 is applied, adding XSI shading to text relating to *NLSPATH*.

NAME

get — get a version of an SCCS file (**DEVELOPMENT**)

SYNOPSIS

XSI

```
get [-begkmnlLpst] [-c cutoff] [-i list] [-r SID] [-x list] file...
```

DESCRIPTION

The *get* utility shall generate a text file from each named SCCS *file* according to the specifications given by its options.

The generated text shall normally be written into a file called the **g-file** whose name is derived from the SCCS filename by simply removing the leading `"s."`.

OPTIONS

The *get* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−r *SID* Indicate the SCCS Identification String (SID) of the version (delta) of an SCCS file to be retrieved. The table shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by *delta* if the **−e** option is also used), as a function of the SID specified.

−c *cutoff* Indicate the *cutoff* date-time, in the form:

YY[*MM*[*DD*[*HH*[*MM*[*SS*]]]]]

For the *YY* component, values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.

Note: It is expected that in a future version of this standard the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)

No changes (deltas) to the SCCS file that were created after the specified *cutoff* date-time shall be included in the generated text file. Units omitted from the date-time default to their maximum possible values; for example, **−c** 7502 is equivalent to **−c** 750228235959.

Any number of non-numeric characters may separate the various 2-digit pieces of the *cutoff* date-time. This feature allows the user to specify a *cutoff* date in the form: **−c** "77/2/2 9:22:25".

−e Indicate that the *get* is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of *delta*. The **−e** option used in a *get* for a particular version (SID) of the SCCS file shall prevent further *get* commands from editing on the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file. Concurrent use of *get* **−e** for different SIDs is always allowed.

If the **g-file** generated by *get* with a **−e** option is accidentally ruined in the process of editing, it may be regenerated by re-executing the *get* command with the **−k** option in place of the **−e** option.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file shall be enforced when the **−e** option is used.

−b Use with the **−e** option to indicate that the new delta should have an SID in a new branch as shown in the table below. This option shall be ignored if the **b** flag is not present in the file or if the retrieved delta is not a leaf delta. (A leaf delta is one that has no successors on the SCCS file tree.)

99083 **Note:** A branch delta may always be created from a non-leaf delta.

99084 **–i** *list* Indicate a *list* of deltas to be included (forced to be applied) in the creation of the
99085 generated file. The *list* has the following syntax:

99086 `<list> ::= <range> | <list> , <range>`
99087 `<range> ::= SID | SID - SID`

99088 SID, the SCCS Identification of a delta, may be in any form shown in the ``SID
99089 Specified'' column of the table in the EXTENDED DESCRIPTION section, except
99090 that the result of supplying a partial SID is unspecified. A diagnostic message shall
99091 be written if the first SID in the range is not an ancestor of the second SID in the
99092 range.

99093 **–x** *list* Indicate a *list* of deltas to be excluded (forced not to be applied) in the creation of
99094 the generated file. See the **–i** option for the *list* format.

99095 **–k** Suppress replacement of identification keywords (see below) in the retrieved text
99096 by their value. The **–k** option shall be implied by the **–e** option.

99097 **–l** Write a delta summary into an **l-file**.

99098 **–L** Write a delta summary to standard output. All informative output that normally is
99099 written to standard output shall be written to standard error instead, unless the **–s**
99100 option is used, in which case it shall be suppressed.

99101 **–p** Write the text retrieved from the SCCS file to the standard output. No **g-file** shall
99102 be created. All informative output that normally goes to the standard output shall
99103 go to standard error instead, unless the **–s** option is used, in which case it shall
99104 disappear.

99105 **–s** Suppress all informative output normally written to standard output. However,
99106 fatal error messages (which shall always be written to the standard error) shall
99107 remain unaffected.

99108 **–m** Precede each text line retrieved from the SCCS file by the SID of the delta that
99109 inserted the text line in the SCCS file. The format shall be:

99110 `"%s\t%s", <SID>, <text line>`

99111 **–n** Precede each generated text line with the %**M**% identification keyword value (see
99112 below). The format shall be:

99113 `"%s\t%s", <%M% value>, <text line>`

99114 When both the **–m** and **–n** options are used, the **<***text line***>** shall be replaced by the
99115 **–m** option-generated format.

99116 **–g** Suppress the actual retrieval of text from the SCCS file. It is primarily used to
99117 generate an **l-file**, or to verify the existence of a particular SID.

99118 **–t** Use to access the most recently created (top) delta in a given release (for example,
99119 **–r 1**), or release and level (for example, **–r 1.2**).

99120 **OPERANDS**

99121 The following operands shall be supported:

99122 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *get*
99123 utility shall behave as though each file in the directory were specified as a named
99124 file, except that non-SCCS files (last component of the pathname does not begin
99125 with **s.**) and unreadable files shall be silently ignored.

99126 If exactly one *file* operand appears, and it is `'-'`, the standard input shall be read;
99127 each line of the standard input is taken to be the name of an SCCS file to be
99128 processed. Non-SCCS files and unreadable files shall be silently ignored.

99129 **STDIN**

99130 The standard input shall be a text file used only if the *file* operand is specified as `'-'`. Each line
99131 of the text file shall be interpreted as an SCCS pathname.

99132 **INPUT FILES**

99133 The SCCS files shall be files of an unspecified format.

99134 **ENVIRONMENT VARIABLES**

99135 The following environment variables shall affect the execution of *get*:

99136 *LANG* Provide a default value for the internationalization variables that are unset or null.
99137 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
99138 variables used to determine the values of locale categories.)

99139 *LC_ALL* If set to a non-empty string value, override the values of all the other
99140 internationalization variables.

99141 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
99142 characters (for example, single-byte as opposed to multi-byte characters in
99143 arguments and input files).

99144 *LC_MESSAGES*

99145 Determine the locale that should be used to affect the format and contents of
99146 diagnostic messages written to standard error, and informative messages written
99147 to standard output (or standard error, if the **−p** option is used).

99148 *NLSPATH* Determine the location of messages objects and message catalogs.

99149 *TZ* Determine the timezone in which the times and dates written in the SCCS file are
99150 evaluated. If the *TZ* variable is unset or NULL, an unspecified system default
99151 timezone is used.

99152 **ASYNCHRONOUS EVENTS**

99153 Default.

99154 **STDOUT**

99155 For each file processed, *get* shall write to standard output the SID being accessed and the
99156 number of lines retrieved from the SCCS file, in the following format:

99157 `"%s\n%d lines\n", <SID>, <number of lines>`

99158 If the **−e** option is used, the SID of the delta to be made shall appear after the SID accessed and
99159 before the number of lines generated, in the POSIX locale:

99160 `"%s\nnew delta %s\n%d lines\n", <SID accessed>,`
99161 `<SID to be made>, <number of lines>`

99162 If there is more than one named file or if a directory or standard input is named, each pathname

shall be written before each of the lines shown in one of the preceding formats:

```
"\n%s:\n", <pathname>
```

If the **–L** option is used, a delta summary shall be written following the format specified below for **l-files**.

If the **–i** option is used, included deltas shall be listed following the notation, in the POSIX locale:

```
"Included:\n"
```

If the **–x** option is used, excluded deltas shall be listed following the notation, in the POSIX locale:

```
"Excluded:\n"
```

If the **–p** or **–L** options are specified, the standard output shall consist of the text retrieved from the SCCS file.

STDERR

The standard error shall be used only for diagnostic messages, except if the **–p** or **–L** options are specified, it shall include all informative messages normally sent to standard output.

OUTPUT FILES

Several auxiliary files may be created by *get*. These files are known generically as the **g-file**, **l-file**, **p-file**, and **z-file**. The letter before the **<**hyphen-minus**>** is called the *tag*. An auxiliary filename shall be formed from the SCCS filename: the application shall ensure that the last component of all SCCS filenames is of the form **s.***module-name*; the auxiliary files shall be named by replacing the leading **s** with the tag. The **g-file** shall be an exception to this scheme: the **g-file** is named by removing the **s.** prefix. For example, for **s.xyz.c**, the auxiliary filenames would be **xyz.c**, **l.xyz.c**, **p.xyz.c**, and **z.xyz.c**, respectively.

The **g-file**, which contains the generated text, shall be created in the current directory (unless the **–p** option is used). A **g-file** shall be created in all cases, whether or not any lines of text were generated by the *get*. It shall be owned by the real user. If the **–k** option is used or implied, the **g-file** shall be writable by the owner only (read-only for everyone else); otherwise, it shall be read-only. Only the real user need have write permission in the current directory.

The **l-file** shall contain a table showing which deltas were applied in generating the retrieved text. The **l-file** shall be created in the current directory if the **–l** option is used; it shall be read-only and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the **l-file** shall have the following format:

```
"%c%c%cΔ%s\t%sΔ%s\n", <code1>, <code2>, <code3>,
    <SID>, <date-time>, <login>
```

where the entries are:

<*code1***>** A **<**space**>** if the delta was applied; `'*'` otherwise.

<*code2***>** A **<**space**>** if the delta was applied or was not applied and ignored; `'*'` if the delta was not applied and was not ignored.

<*code3***>** A character indicating a special reason why the delta was or was not applied:

I Included.

X Excluded.

C Cut off (by a –**c** option).

<*date-time***>** Date and time (using the format of the *date* utility's `%y/%m/%d %T` conversion specification format) of creation.

<*login***>** Login name of person who created *delta*.

The comments and MR data shall follow on subsequent lines, indented one **<**tab**>**. A blank line shall terminate each entry.

The **p-file** shall be used to pass information resulting from a *get* with a –**e** option along to *delta*. Its contents shall also be used to prevent a subsequent execution of *get* with a –**e** option for the same SID until *delta* is executed or the joint edit flag, **j**, is set in the SCCS file. The **p-file** shall be created in the directory containing the SCCS file and the application shall ensure that the effective user has write permission in that directory. It shall be writable by owner only, and owned by the effective user. Each line in the **p-file** shall have the following format:

```
"%sΔ%sΔ%sΔ%s%s%s\n", <g-file SID>,
    <SID of new delta>, <login-name of real user>,
    <date-time>, <i-value>, <x-value>
```

where **<***i-value***>** uses the format `""` if no –**i** option was specified, and shall use the format:

```
"Δ-i%s", <-i option option-argument>
```

if a –**i** option was specified and **<***x-value***>** uses the format `""` if no –**x** option was specified, and shall use the format:

```
"Δ-x%s", <-x option option-argument>
```

if a –**x** option was specified. There can be an arbitrary number of lines in the **p-file** at any time; no two lines shall have the same new delta SID.

The **z-file** shall serve as a lock-out mechanism against simultaneous updates. Its contents shall be the binary process ID of the command (that is, *get*) that created it. The **z-file** shall be created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the **p-file** shall apply for the **z-file**. The **z-file** shall be created read-only.

EXTENDED DESCRIPTION

Determination of SCCS Identification String				
SID* Specified	**–b Keyletter Used†**	**Other Conditions**	**SID Retrieved**	**SID of Delta to be Created**
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	–	R < mR and R does not exist	hR.mL**	hR.mL.(mB+1).1
R	–	Trunk successor in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk successor	R.L	R.(L+1)
R.L	yes	No trunk successor	R.L	R.L.(mB+1).1
R.L	–	Trunk successor in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch successor	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch successor	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch successor	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch successor	R.L.B.S	R.L.(mB+1).1
R.L.B.S	–	Branch successor	R.L.B.S	R.L.(mB+1).1

* R, L, B, and S are the release, level, branch, and sequence components of the SID, respectively; m means maximum. Thus, for example, R.mL means ``the maximum level number within release R''; R.L.(mB+1).1 means ``the first sequence number on the new branch (that is, maximum branch number plus one) of level L within release R''. Note that if the SID specified is of the form R.L, R.L.B, or R.L.B.S, each of the specified components shall exist.

** hR is the highest existing release that is lower than the specified, nonexistent, release R.

*** This is used to force creation of the first delta in a new release.

† The **–b** option is effective only if the **b** flag is present in the file. An entry of `'–'` means ``irrelevant''.

‡ This case applies if the **d** (default SID) flag is not present in the file. If the **d** flag is present in the file, then the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

99267 **System Date and Time**

99268 When a **g-file** is generated, the creation time of deltas in the SCCS file may be taken into
99269 account. If any of these times are apparently in the future, the behavior is unspecified.

99270 **Identification Keywords**

99271 Identifying information shall be inserted into the text retrieved from the SCCS file by replacing
99272 identification keywords with their value wherever they occur. The following keywords may be
99273 used in the text stored in an SCCS file:

99274 %**M**% Module name: either the value of the **m** flag in the file, or if absent, the name of the
99275 SCCS file with the leading **s.** removed.

99276 %**I**% SCCS identification (SID) (%**R**%.%**L**% or %**R**%.%**L**%.%**B**%.%**S**%) of the retrieved
99277 text.

99278 %**R**% Release.

99279 %**L**% Level.

99280 %**B**% Branch.

99281 %**S**% Sequence.

99282 %**D**% Current date (*YY*/*MM*/*DD*).

99283 %**H**% Current date (*MM*/*DD*/*YY*).

99284 %**T**% Current time (*HH*:*MM*:*SS*).

99285 %**E**% Date newest applied delta was created (*YY*/*MM*/*DD*).

99286 %**G**% Date newest applied delta was created (*MM*/*DD*/*YY*).

99287 %**U**% Time newest applied delta was created (*HH*:*MM*:*SS*).

99288 %**Y**% Module type: value of the **t** flag in the SCCS file.

99289 %**F**% SCCS filename.

99290 %**P**% SCCS absolute pathname.

99291 %**Q**% The value of the **q** flag in the file.

99292 %**C**% Current line number. This keyword is intended for identifying messages output by
99293 the program, such as ``this should not have happened'' type errors. It is not
99294 intended to be used on every line to provide sequence numbers.

99295 %**Z**% The four-character string `"@(#)"` recognizable by *what*.

99296 %**W**% A shorthand notation for constructing *what* strings:

99297 `%W%=%Z%%M%<tab>%I%`

99298 %**A**% Another shorthand notation for constructing *what* strings:

99299 `%A%=%Z%%Y%%M%%I%%Z%`

99300 **EXIT STATUS**

99301 The following exit values shall be returned:

99302 0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Problems can arise if the system date and time have been modified (for example, put forward and then back again, or unsynchronized clocks across a network) and can also arise when different values of the *TZ* environment variable are used.

Problems of a similar nature can also arise for the operation of the *delta* utility, which compares the previous file body against the working file as part of its normal operation.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

admin, *delta*, *prs*, *what*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 5

A correction is made to the first format string in STDOUT.

The interpretation of the *YY* component of the **–c** *cutoff* argument is noted.

Issue 6

The obsolescent SYNOPSIS is removed, removing the **–lp** option.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

The Open Group Corrigendum U025/5 is applied, correcting text in the OPTIONS section.

The Open Group Corrigendum U048/1 is applied.

The Open Group Interpretation PIN4C.00014 is applied.

The Open Group Base Resolution bwg2001-007 is applied as follows:

- The EXTENDED DESCRIPTION section is updated to make partial SID handling unspecified, reflecting common usage, and to clarify SID ranges.
- New text is added to the EXTENDED DESCRIPTION and APPLICATION USAGE sections regarding how the system date and time may be taken into account.

- The *TZ* environment variable is added to the ENVIRONMENT VARIABLES section.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0104 [584] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to behave as follows:

a. Report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

b. Disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

getconf — get configuration values

SYNOPSIS

```
getconf [-v specification] system_var
getconf [-v specification] path_var pathname
```

DESCRIPTION

In the first synopsis form, the *getconf* utility shall write to the standard output the value of the variable specified by the *system_var* operand.

In the second synopsis form, the *getconf* utility shall write to the standard output the value of the variable specified by the *path_var* operand for the path specified by the *pathname* operand.

The value of each configuration variable shall be determined as if it were obtained by calling the function from which it is defined to be available by this volume of POSIX.1-2024 or by the System Interfaces volume of POSIX.1-2024 (see the OPERANDS section). The value shall reflect conditions in the current operating environment.

OPTIONS

The *getconf* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

−v *specification*

Indicate a specific specification and version for which configuration variables shall be determined. If this option is not specified, the values returned correspond to an implementation default conforming compilation environment.

If the command:

```
getconf _POSIX_V8_ILP32_OFF32
```

does not write `"-1\n"` or `"undefined\n"` to standard output, then commands of the form:

```
getconf -v POSIX_V8_ILP32_OFF32 ...
```

determine values for configuration variables corresponding to the POSIX_V8_ILP32_OFF32 compilation environment specified in *c17*, the EXTENDED DESCRIPTION.

If the command:

```
getconf _POSIX_V8_ILP32_OFFBIG
```

does not write `"-1\n"` or `"undefined\n"` to standard output, then commands of the form:

```
getconf -v POSIX_V8_ILP32_OFFBIG ...
```

determine values for configuration variables corresponding to the POSIX_V8_ILP32_OFFBIG compilation environment specified in *c17*, the EXTENDED DESCRIPTION.

If the command:

```
getconf _POSIX_V8_LP64_OFF64
```

does not write `"-1\n"` or `"undefined\n"` to standard output, then commands of the form:


```
getconf -v POSIX_V8_LP64_OFF64 ...
```

determine values for configuration variables corresponding to the POSIX_V8_LP64_OFF64 compilation environment specified in *c17*, the EXTENDED DESCRIPTION.

If the command:

```
getconf _POSIX_V8_LPBIG_OFFBIG
```

does not write `"-1\n"` or `"undefined\n"` to standard output, then commands of the form:

```
getconf -v POSIX_V8_LPBIG_OFFBIG ...
```

determine values for configuration variables corresponding to the POSIX_V8_LPBIG_OFFBIG compilation environment specified in *c17*, the EXTENDED DESCRIPTION.

OPERANDS

The following operands shall be supported:

path_var A name of a configuration variable. All of the variables in the Variable column of the table in the DESCRIPTION of the *fpathconf*( ) function defined in the System Interfaces volume of POSIX.1-2024, without the enclosing braces, shall be supported. The implementation may add other local variables.

pathname A pathname for which the variable specified by *path_var* is to be determined.

system_var A name of a configuration variable. All of the following variables shall be supported:

- The names, without the enclosing braces, in the Variable column of the table in the DESCRIPTION of the *sysconf*( ) function in the System Interfaces volume of POSIX.1-2024, except for the entries corresponding to _SC_CLK_TCK, _SC_GETGR_R_SIZE_MAX, _SC_GETPW_R_SIZE_MAX, _SC_NPROCESSORS_CONF, _SC_NPROCESSORS_ONLN, and _SC_NSIG.

  For compatibility with earlier versions, the following variable names shall also be supported:

  POSIX2_C_BIND
  POSIX2_C_DEV
  POSIX2_CHAR_TERM
  POSIX2_FORT_RUN
  POSIX2_LOCALEDEF
  POSIX2_SW_DEV
  POSIX2_UPE
  POSIX2_VERSION

  and shall be equivalent to the same name prefixed with an <underscore>. This requirement may be removed in a future version.

- The names NPROCESSORS_CONF and NPROCESSORS_ONLN. The values of these configuration variables shall be determined as if they were obtained by calling the function *sysconf*( ) with the argument _SC_NPROCESSORS_CONF or _SC_NPROCESSORS_ONLN, respectively.

- The names of the symbolic constants used as the *name* argument of the *confstr*( ) function in the System Interfaces volume of POSIX.1-2024, without the _CS_ prefix.
- The names of the symbolic constants listed under the headings ``Maximum Values'' and ``Minimum Values'' in the description of the **<limits.h>** header in the Base Definitions volume of POSIX.1-2024, without the enclosing braces.

  For compatibility with earlier versions, the following variable names shall also be supported:

  POSIX2_BC_BASE_MAX
  POSIX2_BC_DIM_MAX
  POSIX2_BC_SCALE_MAX
  POSIX2_BC_STRING_MAX
  POSIX2_COLL_WEIGHTS_MAX
  POSIX2_EXPR_NEST_MAX
  POSIX2_LINE_MAX
  POSIX2_RE_DUP_MAX

  and shall be equivalent to the same name prefixed with an <underscore>. This requirement may be removed in a future version.

The implementation may add other local values.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *getconf*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

99478 **STDOUT**

99479 If the specified variable is defined on the system and its value is described to be available from
99480 the *confstr*( ) function defined in the System Interfaces volume of POSIX.1-2024, its value shall be
99481 written in the following format:

99482 `"%s\n", <value>`

99483 Otherwise, if the specified variable is defined on the system, its value shall be written in the
99484 following format:

99485 `"%d\n", <value>`

99486 If the specified variable is valid, but is undefined on the system, *getconf* shall write using the
99487 following format:

99488 `"undefined\n"`

99489 If the variable name is invalid or an error occurs, nothing shall be written to standard output.

99490 **STDERR**

99491 The standard error shall be used only for diagnostic messages.

99492 **OUTPUT FILES**

99493 None.

99494 **EXTENDED DESCRIPTION**

99495 None.

99496 **EXIT STATUS**

99497 The following exit values shall be returned:

99498 0 The specified variable is valid and information about its current state was written
99499 successfully.

99500 >0 An error occurred.

99501 **CONSEQUENCES OF ERRORS**

99502 Default.

99503 **APPLICATION USAGE**

99504 None.

99505 **EXAMPLES**

99506 The following example illustrates the value of {NGROUPS_MAX}:

99507 ```
getconf NGROUPS_MAX
```

99508 The following example illustrates the value of {NAME_MAX} for a specific directory:

99509 ```
getconf NAME_MAX /usr
```

99510 The following example shows how to deal more carefully with results that might be unspecified:

99511 ```
if value=$(getconf PATH_MAX /usr); then
99512   if ["$value" = "undefined"]; then
99513   echo PATH_MAX in /usr is indeterminate.
99514   else
99515   echo PATH_MAX in /usr is $value.
99516   fi
99517 else
99518   echo Error in getconf.
99519 fi
```

  
```

RATIONALE

The original need for this utility, and for the *confstr*( ) function, was to provide a way of finding the configuration-defined default value for the *PATH* environment variable. Since *PATH* can be modified by the user to include directories that could contain utilities replacing the standard utilities, shell scripts need a way to determine the system-supplied *PATH* environment variable value that contains the correct search path for the standard utilities. It was later suggested that access to the other variables described in this volume of POSIX.1-2024 could also be useful to applications.

This functionality of *getconf* would not be adequately subsumed by another command such as:

```
grep var /etc/conf
```

because such a strategy would provide correct values for neither those variables that can vary at runtime, nor those that can vary depending on the path.

Early proposal versions of *getconf* specified exit status 1 when the specified variable was valid, but not defined on the system. The output string `"undefined"` is now used to specify this case with exit code 0 because so many things depend on an exit code of zero when an invoked utility is successful.

FUTURE DIRECTIONS

None.

SEE ALSO

c17

XBD Chapter 8 (on page 167), Section 12.2 (on page 215), **<limits.h>**

XSH *confstr*( ), *fpathconf*( ), *sysconf*( ), *system*( )

CHANGE HISTORY

First released in Issue 4.

Issue 5

In the OPERANDS section:

- {NL_MAX} is changed to {NL_NMAX}.
- Entries beginning NL_ are deleted from the list of standard configuration variables.
- The list of variables previously marked UX is merged with the list marked EX.
- Operands are added to support new Option Groups.
- Operands are added so that *getconf* can determine supported programming environments.

Issue 6

The Open Group Corrigendum U029/4 is applied, correcting the example command in the last paragraph of the OPTIONS section.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- Operands are added to determine supported programming environments.

This reference page is updated for alignment with the ISO/IEC 9899: 1999 standard. Specifically, new macros for *c99* programming environments are introduced.

XSI marked *system_var* (XBS5_*) values are marked LEGACY.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/27 is applied, correcting the descriptions

of *path_var* and *system_var* in the OPERANDS section.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The EXAMPLES section is corrected.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0091 [125] is applied.

Issue 8

Austin Group Defect 339 is applied, adding the *system_var* names NPROCESSORS_CONF and NPROCESSORS_ONLN.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1330 is applied, removing obsolescent interfaces and changing ``_V7_'' to ``_V8_''.

99572 **NAME**

99573 getopts — parse utility options

99574 **SYNOPSIS**

99575 `getopts` *`optstring name`* **`[`***`param...`***`]`**

99576 **DESCRIPTION**

99577 The *getopts* utility shall retrieve options and option-arguments from a list of parameters. It shall
99578 support the Utility Syntax Guidelines 3 to 10, inclusive, described in XBD Section 12.2 (on page
99579 215).

99580 When the shell is first invoked, the shell variable *OPTIND* shall be initialized to 1. Each time
99581 *getopts* is invoked, it shall place the value of the next option found in the parameter list in the
99582 shell variable specified by the *name* operand and the shell variable *OPTIND* shall be set as
99583 follows:

99584 • When *getopts* successfully parses an option that takes an option-argument (that is, a
99585 character followed by <colon> in *optstring*, and exit status is 0), the value of *OPTIND* shall
99586 be the integer index of the next element of the parameter list (if any; see OPERANDS
99587 below) to be searched for an option character. Index 1 identifies the first element of the
99588 parameter list.

99589 • When *getopts* reports end of options (that is, when exit status is 1), the value of *OPTIND*
99590 shall be the integer index of the next element of the parameter list (if any).

99591 • In all other cases, the value of *OPTIND* is unspecified, but shall encode the information
99592 needed for the next invocation of *getopts* to resume parsing options after the option just
99593 parsed.

99594 When the option requires an option-argument, the *getopts* utility shall place it in the shell
99595 variable *OPTARG*. If no option was found, or if the option that was found does not have an
99596 option-argument, *OPTARG* shall be unset.

99597 If an option character not contained in the *optstring* operand is found where an option character
99598 is expected, the shell variable specified by *name* shall be set to the <question-mark> (`'?'`)
99599 character. In this case, if the first character in *optstring* is a <colon> (`':'`), the shell variable
99600 *OPTARG* shall be set to the option character found, but no output shall be written to standard
99601 error; otherwise, the shell variable *OPTARG* shall be unset and a diagnostic message shall be
99602 written to standard error. This condition shall be considered to be an error detected in the way
99603 arguments were presented to the invoking application, but shall not be an error in *getopts*
99604 processing.

99605 If an option-argument is missing:

99606 • If the first character of *optstring* is a <colon>, the shell variable specified by *name* shall be
99607 set to the <colon> character and the shell variable *OPTARG* shall be set to the option
99608 character found.

99609 • Otherwise, the shell variable specified by *name* shall be set to the <question-mark>
99610 character, the shell variable *OPTARG* shall be unset, and a diagnostic message shall be
99611 written to standard error. This condition shall be considered to be an error detected in the
99612 way arguments were presented to the invoking application, but shall not be an error in
99613 *getopts* processing; a diagnostic message shall be written as stated, but the exit status shall
99614 be zero.

99615 When the end of options is encountered, the *getopts* utility shall exit with a return value of one;
99616 the shell variable *OPTIND* shall be set to the index of the argument containing the first operand
99617 in the parameter list, or the value 1 plus the number of elements in the parameter list if there are

no operands in the parameter list; the *name* variable shall be set to the <question-mark> character. Any of the following shall identify the end of options: the first `"--"` element of the parameter list that is not an option-argument, finding an element of the parameter list that is not an option-argument and does not begin with a `'-'`, or encountering an error.

The shell variables *OPTIND* and *OPTARG* shall not be exported by default. An error in setting any of these variables (such as if *name* has previously been marked *readonly*) shall be considered an error of *getopts* processing, and shall result in a return value greater than one.

The *getopts* utility can affect *OPTIND*, *OPTARG*, and the shell variable specified by the *name* operand, within the current shell execution environment; see Section 2.13 (on page 2522).

If the application sets *OPTIND* to the value 1, a new set of parameters can be used: either the current positional parameters or new *param* values. Any other attempt to invoke *getopts* multiple times in a single shell execution environment with parameters (positional parameters or *param* operands) that are not the same in all invocations, or with an *OPTIND* value modified by the application to be a value other than 1, produces unspecified results.

OPTIONS

None.

OPERANDS

The following operands shall be supported:

optstring A string containing the option characters recognized by the utility invoking *getopts*. If a character is followed by a <colon>, the option shall be expected to have an argument, which should be supplied as a separate argument. Applications should specify an option character and its option-argument as separate arguments, but *getopts* shall interpret the characters following an option character requiring arguments as an argument whether or not this is done. An explicit null option-argument need not be recognized if it is not supplied as a separate argument when *getopts* is invoked. (See also the *getopt*( ) function defined in the System Interfaces volume of POSIX.1-2024.) The characters <question-mark> and <colon> shall not be used as option characters by an application. The use of other option characters that are not alphanumeric produces unspecified results. Whether or not the option-argument is supplied as a separate argument from the option character, the value in *OPTARG* shall only be the characters of the option-argument. The first character in *optstring* determines how *getopts* behaves if an option character is not known or an option-argument is missing.

name The name of a shell variable that shall be set by the *getopts* utility to the option character that was found.

By default, the list of parameters parsed by the *getopts* utility shall be the positional parameters currently set in the invoking shell environment (`"$@"`). If *param* operands are given, they shall be parsed instead of the positional parameters. Note that the next element of the parameter list need not exist; in this case, *OPTIND* will be set to `$#+1` or the number of *param* operands plus 1.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *getopts*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

OPTIND This variable shall be used by the *getopts* utility as the index of the next argument to be processed.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

Whenever an error is detected and the first character in the *optstring* operand is not a <colon> (`':'`), a diagnostic message shall be written to standard error with the following information in an unspecified format:

- The invoking program name shall be identified in the message. The invoking program name shall be the value of the shell special parameter 0 (see Section 2.5.2, on page 2479) at the time the *getopts* utility is invoked. A name equivalent to:

  ```
  basename "$0"
  ```

  may be used.
- If an option is found that was not specified in *optstring*, this error is identified and the invalid option character shall be identified in the message.
- If an option requiring an option-argument is found, but an option-argument is not found, this error shall be identified and the invalid option character shall be identified in the message.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 An option, specified or unspecified by *optstring*, was found.

99702 1 The end of options was encountered.

99703 >1 An error occurred.

99704 **CONSEQUENCES OF ERRORS**

99705 Default.

99706 **APPLICATION USAGE**

99707 This utility is required to be intrinsic. See Section 1.7 (on page 2470) for details.

99708 Since *getopts* affects the current shell execution environment, it is generally provided as a shell
99709 regular built-in. If it is called in a subshell or separate utility execution environment, such as one
99710 of the following:

```
99711 (getopts abc value "$@")
99712 nohup getopts ...
99713 find . -exec getopts ... \;
```

99714 it does not affect the shell variables in the caller's environment.

99715 Note that shell functions share *OPTIND* with the calling shell even though the positional
99716 parameters are changed. If the calling shell and any of its functions uses *getopts* to parse
99717 arguments, the results are unspecified.

99718 **EXAMPLES**

99719 The following example script parses and displays its arguments:

```
99720 aflag=
99721 bflag=
99722 while getopts ab: name
99723 do
99724     case $name in
99725     a)    aflag=1;;
99726     b)    bflag=1
99727           bval="$OPTARG";;
99728     ?)   printf "Usage: %s: [-a] [-b value] args\n" $0
99729           exit 2;;
99730     esac
99731 done
99732 if [ -n "$aflag" ]; then
99733     printf "Option -a specified\n"
99734 fi
99735 if [ -n "$bflag" ]; then
99736     printf 'Option -b "%s" specified\n' "$bval"
99737 fi
99738 shift $(($OPTIND - 1))
99739 printf "Remaining arguments are: %s\n" "$*"
```

99740 **RATIONALE**

99741 The *getopts* utility was chosen in preference to the System V *getopt* utility because *getopts* handles
99742 option-arguments containing <blank> characters.

99743 The *OPTARG* variable is not mentioned in the ENVIRONMENT VARIABLES section because it
99744 does not affect the execution of *getopts*; it is one of the few ``output-only'' variables used by the
99745 standard utilities.

99746 The <colon> is not allowed as an option character because that is not historical behavior, and it
99747 violates the Utility Syntax Guidelines. The <colon> is now specified to behave as in the

99748 KornShell version of the *getopts* utility; when used as the first character in the *optstring* operand,
99749 it disables diagnostics concerning missing option-arguments and unexpected option characters.
99750 This replaces the use of the *OPTERR* variable that was specified in an early proposal.

99751 Although a leading <plus-sign> in *optstring* is required to have no effect on the behavior of
99752 *getopt*( ), this standard intentionally allows implementations of the *getopts* utility to use a leading
99753 <plus-sign> as an extension that alters behavior. In fact, a <plus-sign> anywhere in the *optstring*
99754 in the *getopts* utility produces unspecified behavior.

99755 The formats of the diagnostic messages produced by the *getopts* utility and the *getopt*( ) function
99756 are not fully specified because implementations with superior (``friendlier'') formats objected to
99757 the formats used by some historical implementations. The standard developers considered it
99758 important that the information in the messages used be uniform between *getopts* and *getopt*( ).
99759 Exact duplication of the messages might not be possible, particularly if a utility is built on
99760 another system that has a different *getopt*( ) function, but the messages must have specific
99761 information included so that the program name, invalid option character, and type of error can
99762 be distinguished by a user.

99763 Only a rare application program intercepts a *getopts* standard error message and wants to parse
99764 it. Therefore, implementations are free to choose the most usable messages they can devise. The
99765 following formats are used by many historical implementations:

99766 `"%s: illegal option -- %c\n", <program name>, <option character>`

99767 `"%s: option requires an argument -- %c\n", <program name>, \`
99768 `    <option character>`

99769 Historical shells with built-in versions of *getopt*( ) or *getopts* have used different formats,
99770 frequently not even indicating the option character found in error.

99771 **FUTURE DIRECTIONS**

99772 None.

99773 **SEE ALSO**

99774 Section 2.5.2 (on page 2479)

99775 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

99776 XSH *getopt*( )

99777 **CHANGE HISTORY**

99778 First released in Issue 4.

99779 **Issue 6**

99780 The normative text is reworded to avoid use of the term ``must'' for application requirements.

99781 **Issue 7**

99782 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0092 [159] is applied.

99783 **Issue 8**

99784 Austin Group Defect 191 is applied, adding a paragraph about leading <plus-sign> to the
99785 RATIONALE section.

99786 Austin Group Defect 367 is applied, requiring that *getopts* distinguishes between encountering
99787 the end of options and an error occurring, setting its exit status to one and greater than one,
99788 respectively.

99789 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that
99790 this utility is required to be intrinsic.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1442 is applied, changing the EXAMPLES section.

Austin Group Defect 1784 is applied, clarifying several aspects of *getopts* behavior and changing the value of *OPTIND* to be unspecified in some circumstances.

NAME

gettext, ngettext — retrieve text string from messages object

SYNOPSIS

```
gettext [-e|-E] [-d textdomain] [textdomain] msgid
gettext [-e|-E] [-n] -s [-d textdomain] msgid...
ngettext [-e|-E] [-d textdomain] [textdomain] msgid msgid_plural n
```

DESCRIPTION

The *gettext* and *ngettext* utilities shall write to standard output the message string(s) that would result from the following calls to functions defined in the System Interfaces volume of POSIX.1-2024:

```
if (textdomainname == NULL || textdomainname[0] == '\0')
    message_string = msgid;
else {
    setlocale(LC_ALL, "");
    if (textdomaindir != NULL)
       bindtextdomain(textdomainname, textdomaindir);
    if (msgid_plural == NULL)
       message_string = dgettext(textdomainname, msgid);
    else
       message_string = dngettext(textdomainname, msgid, msgid_plural, n);
}
```

where:

- The *textdomaindir* variable is a string containing the value of the *TEXTDOMAINDIR* environment variable, if set and not empty, or is NULL otherwise.
- The *textdomainname* variable is a string containing the text domain name obtained from, in decreasing order of precedence:
  - The optional operand *textdomain,* if present
  - The **–d** *textdomain* option, if specified
  - The *TEXTDOMAIN* environment variable, if set and not empty

  If the text domain name cannot be obtained from these sources, the *textdomainname* variable is NULL.
- If the **–s** option of *gettext* is not specified and for the *ngettext* utility, the *msgid* variable is a string containing:
  - The value of the *msgid* operand, if the **–E** option is specified
  - The value of the *msgid* operand with C-language escape sequences processed (see below), if the **–e** option is specified
  - The value of the *msgid* operand with C-language escape sequences optionally processed (see below), otherwise
- If the **–s** option of *gettext* is specified, the *msgid* variable is a string containing:
  - The value of each *msgid* operand in turn, if the **–E** option is specified or neither the **–e** nor the **–E** option is specified

99836 — The value of each *msgid* operand in turn with C-language escape sequences
99837 processed (see below), if the **−e** option is specified

99838 • For the *gettext* utility, the *msgid_plural* variable is NULL. For the *ngettext* utility, the
99839 *msgid_plural* variable is a string containing:

99840 — The value of the *msgid_plural* operand, if the **−E** option is specified

99841 — The value of the *msgid_plural* operand with C-language escape sequences processed
99842 (see below), if the **−e** option is specified

99843 — The value of the *msgid_plural* operand with C-language escape sequences optionally
99844 processed (see below), otherwise

99845 • For the *gettext* utility, the *n* variable is 1 (one). For the *ngettext* utility the *n* variable is the *n*
99846 operand, parsed as an integer as if by using the *strtoul*( ) function with a *base* argument of
99847 10.

99848 When C-language escape sequences are processed, they shall be processed as specified for
99849 character string literals in the ISO C standard, except that *universal-character-name* escape
99850 sequences need not be supported. Implementations may also support a **<**backslash**>** `'c'` escape
99851 sequence; if supported, the `'\c'` and all characters following it shall be removed and, if the **−s**
99852 option is specified, the behavior shall be as if the **−n** option is also specified.

99853 For the *ngettext* utility, and for the *gettext* utility if the **−s** option is not specified, the resulting
99854 message string shall be written to standard output. If the **−s** option of *gettext* is specified, the
99855 resulting message string for each *msgid* shall be written to standard output with consecutive
99856 message strings separated by a single **<**space**>** character and, if the **−n** option is not specified, a
99857 **<**newline**>** shall be written after the last message string. If the **−s** and **−n** options are specified,
99858 the trailing **<**newline**>** shall be omitted.

99859 Under conditions where the *textdomainname* variable in the above code would be NULL, these
99860 utilities may write a diagnostic message to standard error and exit with non-zero status.

99861 **OPTIONS**

99862 These utilities shall conform to XBD Section 12.2 (on page 215).

99863 The following options shall be supported:

99864 **−d** *textdomain*
99865 Retrieve the translated message from the domain *textdomain*, if *textdomain* is not
99866 specified as an operand.

99867 **−e** Process C-language escape sequences in *msgid* and *msgid_plural* operands.

99868 **−E** Do not process C-language escape sequences in *msgid* and *msgid_plural* operands.

99869 The *gettext* utility shall also support the following options:

99870 **−n** Modify the behavior of the **−s** option such that a **<**newline**>** is not appended to the
99871 output.

99872 **−s** Separate the message strings obtained from each *msgid* operand with **<**space**>**
99873 characters in the output, and (if **−n** is not also specified) append a **<**newline**>** to the
99874 output.

99875 If neither of the mutually exclusive **−e** and **−E** options is specified, it is unspecified which is the
99876 default, except that if the **−s** option of *gettext* is specified then **−E** shall be the default.

99877 **OPERANDS**

99878 The following operands shall be supported:

99879 *textdomain* A text domain name used to retrieve the translated message. This shall override
99880 the specification by the **–d** option, if present.

99881 *msgid* A key to retrieve the translated message.

99882 *msgid_plural* A default plural if no corresponding plural message can be found.

99883 *n* A non-negative decimal integer to be used as the *n* argument to *dngettext*( ) (see the
99884 DESCRIPTION).

99885 **STDIN**

99886 Not used.

99887 **INPUT FILES**

99888 The input files are messages object files (see *msgfmt*).

99889 **ENVIRONMENT VARIABLES**

99890 The following environment variables shall affect the execution of *gettext* and *ngettext*:

99891 *LANG* Provide a default value for the internationalization variables that are unset or null.
99892 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
99893 variables used to determine the values of locale categories.)

99894 XSI *LANGUAGE* Determine the location of messages objects if *NLSPATH* is not set or the evaluation
99895 of *NLSPATH* did not lead to a suitable messages object being found.

99896 *LC_ALL* If set to a non-empty string value, override the values of all the other
99897 internationalization variables.

99898 *LC_MESSAGES*
99899 Determine the locale name used to locate messages objects, and the locale that
99900 should be used to affect the format and contents of diagnostic messages written to
99901 standard error.

99902 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

99903 *TEXTDOMAIN*
99904 Specify the text domain name. (See XBD Section 3.386 (on page 88).)

99905 *TEXTDOMAINDIR*
99906 XSI Specify the pathname to the messages object hierarchy. *NLSPATH* shall have
99907 precedence over *TEXTDOMAINDIR*.

99908 **ASYNCHRONOUS EVENTS**

99909 Default.

99910 **STDOUT**

99911 See DESCRIPTION.

99912 **STDERR**

99913 The standard error shall be used only for diagnostic messages.

99914 **OUTPUT FILES**

99915 None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Since it is unspecified which of the **−e** or **−E** options is the default, except when the **−s** option of *gettext* is specified, portable applications need to ensure that **−e**, **−E**, or (for *gettext*) **−s** is specified whenever a *msgid* or *msgid_plural* operand contains, or might contain, a <backslash> character.

Note that, unless the **−s** option of *gettext* is specified without **−n**, the message(s) written to standard output are not followed by a <newline>. (Therefore the output only ends with a <newline> if the last message ends with one.)

Both *msgid* and *msgid_plural* should be properly quoted for the shell.

EXAMPLES

The following examples assume that the following portable messages object source file (dot-po file) has been compiled to a valid file **mail.mo** by the *msgfmt* utility. See the EXTENDED DESCRIPTION section of the *msgfmt* utility for a description of the dot-po file format.

```
msgid ""
msgstr ""
"Content-Type: text/plain; charset=utf-8\n"
"Plural-Forms: nplurals=4; plural=n==1?0: (n>1&&n<=10)?1: (n==0)?2:3;\n"

msgid "recipient"
msgid_plural "recipients"
msgstr[0] "1 recipient"
msgstr[1] "2 to 10 recipients"
msgstr[2] "no recipients"
msgstr[3] "more than 10 recipients"

msgid "%d attachment\n"
msgid_plural "%d attachments\n"
msgstr[0] "1 (%d) attachment\n"
msgstr[1] "2 to 10 (%d) attachments\n"
msgstr[2] "no (%d) attachments\n"
msgstr[3] "more than 10 (%d) attachments\n"
```

They also assume that **mail.mo** is installed in the directory that *gettext* and *ngettext* search for the current locale. See the OPTIONS and ENVIRONMENT VARIABLES sections above and the description of *gettext*( ) for details on how this search is performed.

The command

```
ngettext -d mail recipient recipients 0
```

will write `"no recipients"`.

The command


```
ngettext -d mail recipient recipients 1
```

will write "`1 recipient`".

The command

```
ngettext -d mail recipient recipients 5
```

will write "`2 to 10 recipients`".

The command

```
ngettext -d mail recipient recipients 11
```

will write "`more than 10 recipients`".

The command

```
ngettext -d mail Call Calls 1
```

will write "`Call`". Note that "`Call`" is not in the messages object.

The command

```
ngettext -d mail Call Calls 0
```

will write "`Calls`".

The command

```
ngettext -d mail Call Calls 10
```

will write "`Calls`".

The command

```
ngettext -ed mail "%d attachment\n" "%d attachments\n" 1
```

will write the same as

```
printf "1 (%%d) attachment\n"
```

(i.e. "`1 (%d) attachment`" followed by a <newline> character). The output of *ngettext* can be used as a format string for *printf*.

The command

```
printf "$(ngettext -ed mail "%d attachment\n" "%d attachments\n" 1)" 10
```

will write the same as

```
printf "1 (%d) attachment\n" 10
```

(i.e. "`1 (10) attachment`" followed by a <newline> character).

The command

```
ngettext -e -d mail "\tsubject\n" "\tsubjects\n" 0
```

will write the same as

```
printf "\tsubjects\n"
```

(i.e. a <tab> character, followed by "`subjects`" followed by a <newline> character). Note that "`\tsubject\n`" is not in the messages object.

The command


```
printf "%s\n" "$(ngettext -E -d mail "subject" "subjects" 0)"
```

will write the same as

```
printf "subjects\n"
```

(i.e. `"subjects"` followed by a <newline> character). Note that `"subject"` is not in the messages object.

The command

```
gettext -s -d mail "recipient"
```

will write `"1 recipient"` followed by a <newline> character.

The command

```
gettext -s -n -d mail "recipient"
```

will write `"1 recipient"` without a <newline> character.

RATIONALE

Historical implementations did not support the `'\a'` C-language escape sequence. This standard requires it to be supported for consistency with other utilities that support the table in XBD Chapter 5 (on page 113).

Unlike other standard utilities, the behavior of *gettext* and *ngettext* is not undefined when *NLSPATH* overrides the system default path; see XBD Section 8.2 (on page 169). This is so that applications can use these utilities to obtain message strings from messages objects in other locations. However, it also means that they need to be implemented in such a way that they do not do anything that would result in undefined behavior when they need to write a diagnostic message. In particular, they should not use a string obtained from a message catalog or a messages object as a format string (or should only do so after checking that the string contains the correct conversions).

FUTURE DIRECTIONS

None.

SEE ALSO

msgfmt, *printf*

XBD Chapter 7 (on page 127), Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *gettext*, *iconv*( ), *setlocale*( )

CHANGE HISTORY

First released in Issue 8.

NAME

grep — search a file for a pattern

SYNOPSIS

```
grep [-E|-F] [-c|-l|-q] [-insvx] -e pattern_list
    [-e pattern_list]... [-f pattern_file]... [file...]

grep [-E|-F] [-c|-l|-q] [-insvx] [-e pattern_list]...
    -f pattern_file [-f pattern_file]... [file...]

grep [-E|-F] [-c|-l|-q] [-insvx] pattern_list [file...]
```

DESCRIPTION

The *grep* utility shall search the input files, selecting lines matching one or more patterns; the types of patterns are controlled by the options specified. The patterns are specified by the **–e** option, **–f** option, or the *pattern_list* operand. The *pattern_list*'s value shall consist of one or more patterns separated by <newline> characters; the *pattern_file*'s contents shall consist of one or more patterns terminated by a <newline> character. By default, an input line shall be selected if any pattern, treated as an entire basic regular expression (BRE) as described in XBD Section 9.3 (on page 181), matches any part of the line excluding the terminating <newline>; a null BRE shall match every line. By default, each selected input line shall be written to the standard output.

Regular expression matching shall be based on text lines. Since a <newline> separates or terminates patterns (see the **–e** and **–f** options below), regular expressions cannot contain a <newline>. Similarly, since patterns are matched against individual lines (excluding the terminating <newline> characters) of the input, there is no way for a pattern to match a <newline> found in the input.

OPTIONS

The *grep* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–E Match using extended regular expressions. Treat each pattern specified as an ERE, as described in XBD Section 9.4 (on page 187). If any entire ERE pattern matches some part of an input line excluding the terminating <newline>, the line shall be matched. A null ERE shall match every line.

–F Match using fixed strings. Treat each pattern specified as a string instead of a regular expression. If an input line contains any of the patterns as a contiguous sequence of bytes, the line shall be matched. A null string shall match every line.

–c Write only a count of selected lines to standard output.

–e *pattern_list*
Specify one or more patterns to be used during the search for input. The application shall ensure that patterns in *pattern_list* are separated by a <newline>. A null pattern can be specified by two adjacent <newline> characters in *pattern_list*. Unless the **–E** or **–F** option is also specified, each pattern shall be treated as a BRE, as described in XBD Section 9.3 (on page 181). Multiple **–e** and **–f** options shall be accepted by the *grep* utility. All of the specified patterns shall be used when matching lines, but the order of evaluation is unspecified.

–f *pattern_file*
Read one or more patterns from the file named by the pathname *pattern_file*. Patterns in *pattern_file* shall be terminated by a <newline>. A null pattern can be specified by an empty line in *pattern_file*. Unless the **–E** or **–F** option is also

specified, each pattern shall be treated as a BRE, as described in XBD Section 9.3 (on page 181).

–i Perform pattern matching in a case-insensitive manner; see XBD Section 9.2 (on page 180).

–l (The letter ell.) Write only the names of files containing selected lines to standard output. Pathnames shall be written once per file searched. If the standard input is searched, a pathname of `"(standard input)"` shall be written, in the POSIX locale. In other locales, `"standard input"` may be replaced by something more appropriate in those locales.

–n Precede each output line by its relative line number in the file, each file starting at line 1. The line number counter shall be reset for each file processed.

–q Quiet. Nothing shall be written to the standard output, regardless of matching lines. Exit with zero status if an input line is selected.

–s Suppress the error messages ordinarily written for nonexistent or unreadable files. Other error messages shall not be suppressed.

–v Select lines not matching any of the specified patterns. If the **–v** option is not specified, selected lines shall be those that match any of the specified patterns.

–x Consider only input lines that use all characters in the line excluding the terminating <newline> to match an entire fixed string or regular expression to be matching lines.

OPERANDS

The following operands shall be supported:

pattern_list Specify one or more patterns to be used during the search for input. This operand shall be treated as if it were specified as **–e** *pattern_list*.

file A pathname of a file to be searched for the patterns. If no *file* operands are specified, the standard input shall be used.

STDIN

The standard input shall be used if no *file* operands are specified, and shall be used if a *file* operand is `'–'` and the implementation treats the `'–'` as meaning standard input. Otherwise, the standard input shall not be used. See the INPUT FILES section.

INPUT FILES

The input files shall be text files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *grep*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes within regular expressions.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If the **–l** option is in effect, the following shall be written for each file containing at least one selected input line:

`"%s\n", <file>`

Otherwise, if more than one *file* argument appears, and **–q** is not specified, the *grep* utility shall prefix each output line by:

`"%s:", <file>`

The remainder of each output line shall depend on the other options specified:

- If the **–c** option is in effect, the remainder of each output line shall contain:

  `"%d\n", <count>`

- Otherwise, if **–c** is not in effect and the **–n** option is in effect, the following shall be written to standard output:

  `"%d:", <line number>`

- Finally, the following shall be written to standard output:

  `"%s", <selected-line contents>`

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 One or more lines were selected and the output specified in STDOUT was successfully written to standard output.

1 No lines were selected.

>1 An error occurred.

CONSEQUENCES OF ERRORS

If the **−q** option is specified, the exit status shall be zero if an input line is selected, even if an error was detected. Otherwise, default actions shall be performed.

APPLICATION USAGE

Care should be taken when using characters in *pattern_list* that may also be meaningful to the command interpreter. It is safest to enclose the entire *pattern_list* argument in single-quotes:

```
'...'
```

The **−e** *pattern_list* option has the same effect as the *pattern_list* operand, but is useful when *pattern_list* begins with the <hyphen-minus> delimiter. It is also useful when it is more convenient to provide multiple patterns as separate arguments.

Multiple **−e** and **−f** options are accepted and *grep* uses all of the patterns it is given while matching input text lines. (Note that the order of evaluation is not specified. If an implementation finds a null string as a pattern, it is allowed to use that pattern first, matching every line, and effectively ignore any other patterns.)

The **−q** option provides a means of easily determining whether or not a pattern (or string) exists in a group of files. When searching several files, it provides a performance improvement (because it can quit as soon as it finds the first match) and requires less care by the user in choosing the set of files to supply as arguments (because it exits zero if it finds a match even if *grep* detected an access or read error on earlier *file* operands).

When using *grep* to process pathnames, it is recommended that LC_ALL, or at least LC_CTYPE and LC_COLLATE, are set to POSIX or C in the environment, since pathnames can contain byte sequences that do not form valid characters in some locales, in which case the utility's behavior would be undefined. In the POSIX locale each byte is a valid single-byte character, and therefore this problem is avoided.

EXAMPLES

1. To find all uses of the word `"Posix"` (in any case) in file **text.mm** and write with line numbers:

   ```
   grep −i −n posix text.mm
   ```

2. To find all empty lines in the standard input:

   ```
   grep ^$
   ```

   or:

   ```
   grep −v .
   ```

3. Both of the following commands print all lines containing strings `"abc"` or `"def"` or both:

   ```
   grep −E 'abc|def'
   ```

   ```
   grep −F 'abc
   def'
   ```

4. Both of the following commands print all lines matching exactly `"abc"` or `"def"`:

   ```
   grep −E '^abc$|^def$'
   ```

   ```
   grep −F −x 'abc
   def'
   ```


RATIONALE

This *grep* has been enhanced in an upwards-compatible way to provide the exact functionality of the historical *egrep* and *fgrep* commands as well. It was the clear intention of the standard developers to consolidate the three *grep*s into a single command.

The old *egrep* and *fgrep* commands are likely to be supported for many years to come as implementation extensions, allowing historical applications to operate unmodified.

Historical implementations usually silently ignored all but one of multiply-specified **−e** and **−f** options, but were not consistent as to which specification was actually used.

The **−b** option was omitted from the OPTIONS section because block numbers are implementation-defined.

The System V restriction on using − to mean standard input was omitted.

A definition of action taken when given a null BRE or ERE is specified. This is an error condition in some historical implementations.

The **−l** option previously indicated that its use was undefined when no files were explicitly named. This behavior was historical and placed an unnecessary restriction on future implementations. It has been removed.

The historical BSD *grep* **−s** option practice is easily duplicated by redirecting standard output to **/dev/null**. The **−s** option required here is from System V.

The **−x** option, historically available only with *fgrep*, is available here for all of the non-obsolescent versions.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

sed

XBD Chapter 8 (on page 167), Chapter 9 (on page 179), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The Open Group Corrigendum U029/5 is applied, correcting the SYNOPSIS.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/28 is applied, correcting the examples using the *grep* **−F** option which did not match the normative description of the **−F** option.

Issue 7

Austin Group Interpretation 1003.1-2001 #092 is applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

SD5-XCU-ERN-98 is applied, updating the STDOUT section.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0105 [584] and XCU/TC2-2008/0106 [663] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1031 is applied, changing the description of the **–i** option.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1502 is applied, changing the EXIT STATUS section.

NAME

hash — remember or report utility locations

SYNOPSIS

```
hash [utility...]
hash -r
```

DESCRIPTION

The *hash* utility shall affect the way the current shell environment remembers the locations of utilities found as described in Section 2.9.1.4 (on page 2502). Depending on the arguments specified, it shall add utility locations to its list of remembered locations or it shall purge the contents of the list. When no arguments are specified, it shall report on the contents of the list.

Utilities provided as built-ins to the shell and functions shall not be reported by *hash*.

OPTIONS

The *hash* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

−r Forget all previously remembered utility locations.

OPERANDS

The following operand shall be supported:

utility The name of a utility to be searched for and added to the list of remembered locations. If the search does not find *utility*, it is unspecified whether or not this is treated as an error. If *utility* contains one or more <slash> characters, the results are unspecified.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *hash*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

PATH Determine the location of *utility*, as described in XBD Chapter 8 (on page 167).

100280 **ASYNCHRONOUS EVENTS**

100281 Default.

100282 **STDOUT**

100283 The standard output of *hash* shall be used when no arguments are specified. Its format is
100284 unspecified, but includes the pathname of each utility in the list of remembered locations for the
100285 current shell environment. This list shall consist of those utilities named in previous *hash*
100286 invocations that have been invoked, and may contain those invoked and found through the
100287 normal command search process. This list shall be cleared when the contents of the *PATH*
100288 environment variable are changed.

100289 **STDERR**

100290 The standard error shall be used only for diagnostic messages.

100291 **OUTPUT FILES**

100292 None.

100293 **EXTENDED DESCRIPTION**

100294 None.

100295 **EXIT STATUS**

100296 The following exit values shall be returned:

100297 0 Successful completion.

100298 >0 An error occurred.

100299 **CONSEQUENCES OF ERRORS**

100300 Default.

100301 **APPLICATION USAGE**

100302 This utility is required to be intrinsic. See Section 1.7 (on page 2470) for details.

100303 Since *hash* affects the current shell execution environment, it is always provided as a shell
100304 regular built-in. If it is called in a separate utility execution environment, such as one of the
100305 following:

100306 `nohup hash -r`
100307 `find . -type f -exec hash {} +`

100308 it does not affect the command search process of the caller's environment.

100309 The *hash* utility may be implemented as an alias—for example, *alias* **–t** –, in which case utilities
100310 found through normal command search are not listed by the *hash* command.

100311 The effects of *hash* **–r** can also be achieved portably by resetting the value of *PATH*; in the
100312 simplest form, this can be:

100313 `PATH="$PATH"`

100314 The use of *hash* with *utility* names is unnecessary for most applications, but may provide a
100315 performance improvement on a few implementations; normally, the hashing process is included
100316 by default.

100317 **EXAMPLES**

100318 None.

RATIONALE

None.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

Section 2.9.1.4 (on page 2502)

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 7

The *hash* utility is moved from the XSI option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0093 [241] is applied.

Issue 8

Austin Group Defect 248 is applied, changing a command line in the APPLICATION USAGE section.

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that this utility is required to be intrinsic.

Austin Group Defect 1063 is applied, clarifying that functions are not reported by *hash*, and that the list of remembered locations is cleared when the contents of *PATH* are changed.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1460 is applied, making it explicitly unspecified whether or not *hash* reports an error if it cannot find *utility*.

NAME

head — copy the first part of files

SYNOPSIS

```
head [-c number|-n number] [file...]
```

DESCRIPTION

The *head* utility shall copy its input files to the standard output, ending the output for each file at a designated point.

Copying shall end at the point in the file indicated by the **–c** *number* or **–n** *number* options. The option-argument *number* shall be counted in units of lines or bytes, according to the options **–n** and **–c**. Both line and byte counts start from 1.

OPTIONS

The *head* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–c *number* The first *number* bytes of each input file shall be copied to standard output. The application shall ensure that the *number* option-argument is a positive decimal integer.

–n *number* This option shall be equivalent to **–c** *number*, except that the ending location in the file shall be measured in lines instead of bytes.

When a file contains less than *number* bytes or lines, it shall be copied to standard output in its entirety. This shall not be an error.

If no options are specified, *head* shall act as if **–n 10** had been specified.

OPERANDS

The following operand shall be supported:

file A pathname of an input file. If no *file* operands are specified, the standard input shall be used.

STDIN

The standard input shall be used if no *file* operands are specified, and shall be used if a *file* operand is `'–'` and the implementation treats the `'–'` as meaning standard input. Otherwise, the standard input shall not be used. See the INPUT FILES section.

INPUT FILES

If the **–c** option is specified, the input files can contain arbitrary data; otherwise, the input files shall be text files, but the line length shall not be restricted to {LINE_MAX} bytes.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *head*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

100390 *LC_MESSAGES*
100391 Determine the locale that should be used to affect the format and contents of
100392 diagnostic messages written to standard error.

100393 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

100394 **ASYNCHRONOUS EVENTS**
100395 Default.

100396 **STDOUT**
100397 The standard output shall contain designated portions of the input files.

100398 If multiple *file* operands are specified, *head* shall precede the output for each with the header:

100399 `"\n==> %s <==\n", <pathname>`

100400 except that the first header written shall not include the initial <newline>.

100401 **STDERR**
100402 The standard error shall be used only for diagnostic messages.

100403 **OUTPUT FILES**
100404 None.

100405 **EXTENDED DESCRIPTION**
100406 None.

100407 **EXIT STATUS**
100408 The following exit values shall be returned:

100409 0 Successful completion.

100410 >0 An error occurred.

100411 **CONSEQUENCES OF ERRORS**
100412 Default.

100413 **APPLICATION USAGE**
100414 When using *head* to process pathnames, it is recommended that LC_ALL, or at least LC_CTYPE
100415 and LC_COLLATE, are set to POSIX or C in the environment, since pathnames can contain byte
100416 sequences that do not form valid characters in some locales, in which case the utility's behavior
100417 would be undefined. In the POSIX locale each byte is a valid single-byte character, and therefore
100418 this problem is avoided.

100419 **EXAMPLES**
100420 To write the first ten lines of all files (except those with a leading period) in the directory:

100421 `head -- *`

100422 **RATIONALE**
100423 Although it is possible to simulate *head* with *sed* 10q for a single file, the standard developers
100424 decided that the popularity of *head* on historical BSD systems warranted its inclusion alongside
100425 *tail*.

100426 POSIX.1-2024 version of *head* follows the Utility Syntax Guidelines. The –**n** option was added to
100427 this new interface so that *head* and *tail* would be more logically related. Earlier versions of this
100428 standard allowed a –**number** option. This form is no longer specified by POSIX.1-2024 but may
100429 be present in some implementations.

100430 The *head* and *tail* utilities have not historically been symmetric. For example, this standard only
100431 requires *tail* to support at most one file operand, while *head* must operate on multiple files.

Conversely, this standard requires *tail* to be able to start at a position relative to the start of a file, but *head* need not support stopping at a position relative to the end of the file. Implementations may choose to make *head* and *tail* symmetric as an extension, but applications should not rely on this.

Older implementations of *head* did not support **–c** *number*, but emulating this via `dd ibs=1 count=number` is much less efficient and emulating via `dd obs=pipe_buf | dd ibs=size count=number_of_blocks` is cumbersome, somewhat less efficient, and can only be used if the number of bytes to be copied is a multiple of a suitable block size less than or equal to {PIPE_BUF}.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

dd, *sed*, *tail*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 6

The obsolescent **–number** form is removed.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

The DESCRIPTION is updated to clarify that when a file contains less than the number of lines requested, the entire file is copied to standard output.

Issue 7

Austin Group Interpretations 1003.1-2001 #027 and #092 are applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The APPLICATION USAGE section is removed and the EXAMPLES section is corrected.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0107 [663] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 407 is applied, adding the **–c** *number* option.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

iconv — codeset conversion

SYNOPSIS

```
iconv [-cs] -f frommap -t tomap [file...]
iconv -f fromcode [-cs] [-t tocode] [file...]
iconv -t tocode [-cs] [-f fromcode] [file...]
iconv -l
```

DESCRIPTION

The *iconv* utility shall convert the encoding of characters in *file* from one codeset to another and write the results to standard output.

When the options indicate that charmap files are used to specify the codesets (see OPTIONS), the codeset conversion shall be accomplished by performing a logical join on the symbolic character names in the two charmaps. The implementation need not support the use of charmap files for codeset conversion unless the POSIX2_LOCALEDEF symbol is defined on the system.

OPTIONS

The *iconv* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−c Omit any characters that are invalid in the codeset of the input file from the output. When **−c** is not used, the results of encountering invalid characters in the input stream (either those that are not characters in the codeset of the input file or that have no corresponding character in the codeset of the output file) shall be specified in the system documentation. The presence or absence of **−c** shall not affect the exit status of *iconv*.

−f *fromcodeset*

Identify the codeset of the input file. The implementation shall recognize the following two forms of the *fromcodeset* option-argument:

fromcode The *fromcode* option-argument can not contain a <slash> character. It shall be interpreted as the name of one of the codeset descriptions provided by the implementation in an unspecified format. Valid values of *fromcode* are implementation-defined.

frommap The *frommap* option-argument needs to contain a <slash> character. It shall be interpreted as the pathname of a charmap file as defined in XBD Section 6.4 (on page 121). If the pathname does not represent a valid, readable charmap file, the results are undefined.

If this option is omitted, the codeset of the current locale shall be used.

−l Write all supported *fromcode* and *tocode* values to standard output in an unspecified format.

−s Suppress any messages written to standard error concerning invalid characters. When **−s** is not used, the results of encountering invalid characters in the input stream (either those that are not valid characters in the codeset of the input file or that have no corresponding character in the codeset of the output file) shall be specified in the system documentation. The presence or absence of **−s** shall not affect the exit status of *iconv*.

100511 **–t** *tocodeset* Identify the codeset to be used for the output file. The implementation shall
100512 recognize the following two forms of the *tocodeset* option-argument:

100513 *tocode* The semantics shall be equivalent to the **–f** *fromcode* option.

100514 *tomap* The semantics shall be equivalent to the **–f** *frommap* option.

100515 If this option is omitted, the codeset of the current locale shall be used.

100516 If either **–f** or **–t** represents a charmap file, but the other does not (or is omitted), or both **–f** and
100517 **–t** are omitted, the results are undefined.

100518 **OPERANDS**

100519 The following operand shall be supported:

100520 *file* A pathname of an input file. If no *file* operands are specified, or if a *file* operand is
100521 `'–'`, the standard input shall be used.

100522 **STDIN**

100523 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is `'–'`.

100524 **INPUT FILES**

100525 The input file shall be a text file.

100526 **ENVIRONMENT VARIABLES**

100527 The following environment variables shall affect the execution of *iconv*:

100528 *LANG* Provide a default value for the internationalization variables that are unset or null.
100529 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
100530 variables used to determine the values of locale categories.)

100531 *LC_ALL* If set to a non-empty string value, override the values of all the other
100532 internationalization variables.

100533 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
100534 characters (for example, single-byte as opposed to multi-byte characters in
100535 arguments). During translation of the file, this variable is superseded by the use of
100536 the *fromcode* option-argument.

100537 *LC_MESSAGES*
100538 Determine the locale that should be used to affect the format and contents of
100539 diagnostic messages written to standard error.

100540 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

100541 **ASYNCHRONOUS EVENTS**

100542 Default.

100543 **STDOUT**

100544 When the **–l** option is used, the standard output shall contain all supported *fromcode* and *tocode*
100545 values, written in an unspecified format.

100546 When the **–l** option is not used, the standard output shall contain the sequence of characters
100547 read from the input files, translated to the specified codeset. Nothing else shall be written to the
100548 standard output.

100549 **STDERR**

100550 The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The user must ensure that both charmap files use the same symbolic names for characters the two codesets have in common.

EXAMPLES

The following example converts the contents of file **mail.x400** from the ISO/IEC 6937:2001 standard codeset to the ISO/IEC 8859-1:1998 standard codeset, and stores the results in file **mail.local**:

```
iconv -f IS6937 -t IS8859 mail.x400 > mail.local
```

RATIONALE

The *iconv* utility can be used portably only when the user provides two charmap files as option-arguments. This is because a single charmap provided by the user cannot reliably be joined with the names in a system-provided character set description. The valid values for *fromcode* and *tocode* are implementation-defined and do not have to have any relation to the charmap mechanisms. As an aid to interactive users, the **–l** option was adopted from the Plan 9 operating system. It writes information concerning these implementation-defined values. The format is unspecified because there are many possible useful formats that could be chosen, such as a matrix of valid combinations of *fromcode* and *tocode*. The **–l** option is not intended for shell script usage; conforming applications will have to use charmaps.

The *iconv* utility may support the conversion between ASCII and EBCDIC-based encodings, but is not required to do so. In an XSI-compliant implementation, the *dd* utility is the only method guaranteed to support conversion between these two character sets.

FUTURE DIRECTIONS

None.

SEE ALSO

dd, *gencat*

XBD Section 6.4 (on page 121), Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 3.

Issue 6

This utility has been rewritten to align with the IEEE P1003.2b draft standard. Specifically, the ability to use charmap files for conversion has been added.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/29 is applied, making changes to address inconsistencies with the *iconv*( ) function in the System Interfaces volume of POSIX.1-2024.

Issue 7

Austin Group Interpretation 1003.1-2001 #206 is applied, correcting the *tomap* option.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0094 [291] and XCU/TC1-2008/0095 [291] are applied.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

id — return user identity

SYNOPSIS

```
id [user]
id -G [-n] [user]
id -g [-nr] [user]
id -u [-nr] [user]
```

DESCRIPTION

If no *user* operand is provided, the *id* utility shall write the user and group IDs and the corresponding user and group names of the invoking process to standard output. If the effective and real IDs do not match, both shall be written. If multiple groups are supported by the underlying system (see the description of {NGROUPS_MAX} in the System Interfaces volume of POSIX.1-2024), the supplementary group affiliations of the invoking process shall also be written.

If a *user* operand is provided and the process has appropriate privileges, the user and group IDs of the selected user shall be written. In this case, effective IDs shall be assumed to be identical to real IDs. If the selected user has more than one allowable group membership listed in the group database, these shall be written in the same manner as the supplementary groups described in the preceding paragraph.

OPTIONS

The *id* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−G Output all different group IDs (effective, real, and supplementary) only, using the format `"%u\n"`. If there is more than one distinct group affiliation, output each such affiliation, using the format `" %u"`, before the <newline> is output.

−g Output only the effective group ID, using the format `"%u\n"`.

−n Output the name in the format `"%s"` instead of the numeric ID using the format `"%u"`.

−r Output the real ID instead of the effective ID.

−u Output only the effective user ID, using the format `"%u\n"`.

OPERANDS

The following operand shall be supported:

user The login name for which information is to be written.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *id*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

100643 *LC_ALL* If set to a non-empty string value, override the values of all the other
100644 internationalization variables.

100645 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
100646 characters (for example, single-byte as opposed to multi-byte characters in
100647 arguments).

100648 *LC_MESSAGES*
100649 Determine the locale that should be used to affect the format and contents of
100650 diagnostic messages written to standard error and informative messages written to
100651 standard output.

100652 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

100653 **ASYNCHRONOUS EVENTS**
100654 Default.

100655 **STDOUT**
100656 The following formats shall be used when the *LC_MESSAGES* locale category specifies the
100657 POSIX locale. In other locales, the strings *uid*, *gid*, *euid*, *egid*, and *groups* may be replaced with
100658 more appropriate strings corresponding to the locale.

100659 `"uid=%u(%s) gid=%u(%s)\n", <real user ID>, <user-name>,`
100660 `    <real group ID>, <group-name>`

100661 If the effective and real user IDs do not match, the following shall be inserted immediately
100662 before the `'\n'` character in the previous format:

100663 `" euid=%u(%s)"`

100664 with the following arguments added at the end of the argument list:

100665 `<effective user ID>, <effective user-name>`

100666 If the effective and real group IDs do not match, the following shall be inserted directly before
100667 the `'\n'` character in the format string (and after any addition resulting from the effective and
100668 real user IDs not matching):

100669 `" egid=%u(%s)"`

100670 with the following arguments added at the end of the argument list:

100671 `<effective group-ID>, <effective group name>`

100672 If the process has supplementary group affiliations or the selected user is allowed to belong to
100673 multiple groups, the first shall be added directly before the **<**newline**>** in the format string:

100674 `" groups=%u(%s)"`

100675 with the following arguments added at the end of the argument list:

100676 `<supplementary group ID>, <supplementary group name>`

100677 and the necessary number of the following added after that for any remaining supplementary
100678 group IDs:

100679 `",%u(%s)"`

100680 and the necessary number of the following arguments added at the end of the argument list:

100681 `<supplementary group ID>, <supplementary group name>`

100682 If any of the user ID, group ID, effective user ID, effective group ID, or supplementary/multiple

group IDs cannot be mapped by the system into printable user or group names, the corresponding `"(%s)"` and *name* argument shall be omitted from the corresponding format string.

When any of the options are specified, the output format shall be as described in the OPTIONS section.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Output produced by the **−G** option and by the default case could potentially produce very long lines on systems that support large numbers of supplementary groups. (On systems with user and group IDs that are 32-bit integers and with group names with a maximum of 8 bytes per name, 93 supplementary groups plus distinct effective and real group and user IDs could theoretically overflow the 2 048-byte {LINE_MAX} text file line limit on the default output case. It would take about 186 supplementary groups to overflow the 2 048-byte barrier using *id* **−G**). This is not expected to be a problem in practice, but in cases where it is a concern, applications should consider using *fold* **−s** before post-processing the output of *id*.

EXAMPLES

None.

RATIONALE

The functionality provided by the 4 BSD *groups* utility can be simulated using:

```
id −Gn [ user ]
```

The 4 BSD command *groups* was considered, but it was not included because it did not provide the functionality of the *id* utility of the SVID. Also, it was thought that it would be easier to modify *id* to provide the additional functionality necessary to systems with multiple groups than to invent another command.

The options **−u**, **−g**, **−n**, and **−r** were added to ease the use of *id* with shell commands substitution. Without these options it is necessary to use some preprocessor such as *sed* to select the desired piece of information. Since output such as that produced by:

```
id −u −n
```

is frequently wanted, it seemed desirable to add the options.

100723 **FUTURE DIRECTIONS**

100724 None.

100725 **SEE ALSO**

100726 *fold*, *logname*, *who*

100727 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

100728 XSH *getgid*( ), *getgroups*( ), *getuid*( )

100729 **CHANGE HISTORY**

100730 First released in Issue 2.

100731 **Issue 7**

100732 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

100733 **Issue 8**

100734 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

ipcrm — remove an XSI message queue, semaphore set, or shared memory segment identifier

SYNOPSIS

XSI
```
ipcrm [-q msgid|-Q msgkey|-s semid|-S semkey|-m shmid|-M shmkey]...
```

DESCRIPTION

The *ipcrm* utility shall remove zero or more message queues, semaphore sets, or shared memory segments. The interprocess communication facilities to be removed are specified by the options.

Only a user with appropriate privileges shall be allowed to remove an interprocess communication facility that was not created by or owned by the user invoking *ipcrm*.

OPTIONS

The *ipcrm* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−q *msgid* Remove the message queue identifier *msgid* from the system and destroy the message queue and data structure associated with it.

−m *shmid* Remove the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it shall be destroyed when all processes with the segment attached have either detached the segment or terminated. If the segment is not attached to any process, it shall be destroyed immediately.

−s *semid* Remove the semaphore identifier *semid* from the system and destroy the set of semaphores and data structure associated with it.

−Q *msgkey* Remove the message queue identifier, created with key *msgkey*, from the system and destroy the message queue and data structure associated with it.

−M *shmkey* Remove the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structure associated with it shall be destroyed after the last detach.

−S *semkey* Remove the semaphore identifier, created with key *semkey*, from the system and destroy the set of semaphores and data structure associated with it.

OPERANDS

None.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *ipcrm*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

100776 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
100777 characters (for example, single-byte as opposed to multi-byte characters in
100778 arguments).

100779 *LC_MESSAGES*
100780 Determine the locale that should be used to affect the format and contents of
100781 diagnostic messages written to standard error.

100782 *NLSPATH* Determine the location of messages objects and message catalogs.

100783 **ASYNCHRONOUS EVENTS**
100784 Default.

100785 **STDOUT**
100786 Not used.

100787 **STDERR**
100788 The standard error shall be used only for diagnostic messages.

100789 **OUTPUT FILES**
100790 None.

100791 **EXTENDED DESCRIPTION**
100792 None.

100793 **EXIT STATUS**
100794 The following exit values shall be returned:

100795 0 Successful completion.

100796 >0 An error occurred.

100797 **CONSEQUENCES OF ERRORS**
100798 Default.

100799 **APPLICATION USAGE**
100800 None.

100801 **EXAMPLES**
100802 None.

100803 **RATIONALE**
100804 None.

100805 **FUTURE DIRECTIONS**
100806 None.

100807 **SEE ALSO**
100808 *ipcs*

100809 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

100810 XSH *msgctl*( ), *semctl*( ), *shmctl*( )

100811 **CHANGE HISTORY**
100812 First released in Issue 5.

100813 **Issue 7**
100814 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

100815 **Issue 8**

100816 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

100817 Austin Group Defect 1240 is applied, clarifying the description of the **–m** option.

NAME

ipcs — report XSI interprocess communication facilities status

SYNOPSIS

XSI

```
ipcs [-qms] [-a|-bcopt]
```

DESCRIPTION

The *ipcs* utility shall write information about active interprocess communication facilities.

Without options, information shall be written in short format for message queues, shared memory segments, and semaphore sets that are currently active in the system. Otherwise, the information that is displayed is controlled by the options specified.

OPTIONS

The *ipcs* utility shall conform to XBD Section 12.2 (on page 215).

The *ipcs* utility accepts the following options:

−q Write information about active message queues.

−m Write information about active shared memory segments.

−s Write information about active semaphore sets.

If **−q**, **−m**, or **−s** are specified, only information about those facilities shall be written. If none of these three are specified, information about all three shall be written subject to the following options:

−a Use all print options. (This is a shorthand notation for **−b**, **−c**, **−o**, **−p**, and **−t**.)

−b Write information on maximum allowable size. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.)

−c Write creator's user name and group name; see below.

−o Write information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues, and number of processes attached to shared memory segments.)

−p Write process number information. (Process ID of the last process to send a message and process ID of the last process to receive a message on message queues, process ID of the creating process, and process ID of the last process to attach or detach on shared memory segments.)

−t Write time information. (Time of the last control operation that changed the access permissions for all facilities, time of the last *msgsnd*( ) and *msgrcv*( ) operations on message queues, time of the last *shmat*( ) and *shmdt*( ) operations on shared memory, and time of the last *semop*( ) operation on semaphores.)

OPERANDS

None.

STDIN

Not used.

INPUT FILES

- The group database

- The user database

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *ipcs*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of messages objects and message catalogs.

TZ Determine the timezone for the date and time strings written by *ipcs*. If *TZ* is unset or null, an unspecified default timezone shall be used.

ASYNCHRONOUS EVENTS

Default.

STDOUT

An introductory line shall be written with the format:

`"IPC status from %s as of %s\n", <`*source*`>, <`*date*`>`

where <*source*> indicates the source used to gather the statistics and <*date*> is the information that would be produced by the *date* command when invoked in the POSIX locale.

The *ipcs* utility then shall create up to three reports depending upon the **−q**, **−m**, and **−s** options. The first report shall indicate the status of message queues, the second report shall indicate the status of shared memory segments, and the third report shall indicate the status of semaphore sets.

If the corresponding facility is not installed or has not been used since the last reboot, then the report shall be written out in the format:

`"%s facility not in system.\n", <`*facility*`>`

where <*facility*> is *Message Queue*, *Shared Memory*, or *Semaphore*, as appropriate. If the facility has been installed and has been used since the last reboot, column headings separated by one or more <space> characters and followed by a <newline> shall be written as indicated below followed by the facility name written out using the format:

`"%s:\n", <`*facility*`>`

where <*facility*> is *Message Queues*, *Shared Memory*, or *Semaphores*, as appropriate. On the second and third reports the column headings need not be written if the last column headings written already provide column headings for all information in that report.

The column headings provided in the first column below and the meaning of the information in those columns shall be given in order below; the letters in parentheses indicate the options that shall cause the corresponding column to appear; ``all'' means that the column shall always appear. Each column is separated by one or more <space> characters. Note that these options

100901 only determine what information is provided for each report; they do not determine which
100902 reports are written.

100903 T (all) Type of facility:

100904 `q` Message queue.

100905 `m` Shared memory segment.

100906 `s` Semaphore.

100907 This field is a single character written using the format `%c`.

100908 ID (all) The identifier for the facility entry. This field shall be written using the format
100909 `%d`.

100910 KEY (all) The key used as an argument to *msgget*(), *semget*(), or *shmget*() to create the
100911 facility entry.

100912 **Note:** The key of a shared memory segment is changed to IPC_PRIVATE when the
100913 segment has been removed until all processes attached to the segment
100914 detach it.

100915 This field shall be written using the format `0x%x`.

100916 MODE (all) The facility access modes and flags. The mode shall consist of 11 characters
100917 that are interpreted as follows.

100918 The first character shall be:

100919 `S` If a process is waiting on a *msgsnd*() operation.

100920 – If the above is not true.

100921 The second character shall be:

100922 `R` If a process is waiting on a *msgrcv*() operation.

100923 `C` or – If the associated shared memory segment is to be cleared when the
100924 first attach operation is executed.

100925 – If none of the above is true.

100926 The next nine characters shall be interpreted as three sets of three bits each.
100927 The first set refers to the owner's permissions; the next to permissions of
100928 others in the usergroup of the facility entry; and the last to all others. Within
100929 each set, the first character indicates permission to read, the second character
100930 indicates permission to write or alter the facility entry, and the last character is
100931 a <hyphen-minus> (`'-'`).

100932 The permissions shall be indicated as follows:

100933 *r* If read permission is granted.

100934 *w* If write permission is granted.

100935 *a* If alter permission is granted.

100936 – If the indicated permission is not granted.

100937 The first character following the permissions specifies if there is an alternate or
100938 additional access control method associated with the facility. If there is no
100939 alternate or additional access control method associated with the facility, a
100940 single <space> shall be written; otherwise, another printable character is

100941 written.

100942 OWNER (all) The user name of the owner of the facility entry. If the user name of the owner
100943 is found in the user database, at least the first eight column positions of the
100944 name shall be written using the format `%s`. Otherwise, the user ID of the
100945 owner shall be written using the format `%d`.

100946 GROUP (all) The group name of the owner of the facility entry. If the group name of the
100947 owner is found in the group database, at least the first eight column positions
100948 of the name shall be written using the format `%s`. Otherwise, the group ID of
100949 the owner shall be written using the format `%d`.

100950 The following nine columns shall be only written out for message queues:

100951 CREATOR (**a**,**c**) The user name of the creator of the facility entry. If the user name of the
100952 creator is found in the user database, at least the first eight column positions of
100953 the name shall be written using the format `%s`. Otherwise, the user ID of the
100954 creator shall be written using the format `%d`.

100955 CGROUP (**a**,**c**) The group name of the creator of the facility entry. If the group name of the
100956 creator is found in the group database, at least the first eight column positions
100957 of the name shall be written using the format `%s`. Otherwise, the group ID of
100958 the creator shall be written using the format `%d`.

100959 CBYTES (**a**,**o**) The number of bytes in messages currently outstanding on the associated
100960 message queue. This field shall be written using the format `%d`.

100961 QNUM (**a**,**o**) The number of messages currently outstanding on the associated message
100962 queue. This field shall be written using the format `%d`.

100963 QBYTES (**a**,**b**) The maximum number of bytes allowed in messages outstanding on the
100964 associated message queue. This field shall be written using the format `%d`.

100965 LSPID (**a**,**p**) The process ID of the last process to send a message to the associated queue.
100966 This field shall be written using the format:

100967 `"%d", <`*pid*`>`

100968 where <*pid*> is 0 if no message has been sent to the corresponding message
100969 queue; otherwise, <*pid*> shall be the process ID of the last process to send a
100970 message to the queue.

100971 LRPID (**a**,**p**) The process ID of the last process to receive a message from the associated
100972 queue. This field shall be written using the format:

100973 `"%d", <`*pid*`>`

100974 where <*pid*> is 0 if no message has been received from the corresponding
100975 message queue; otherwise, <*pid*> shall be the process ID of the last process to
100976 receive a message from the queue.

100977 STIME (**a**,**t**) The time the last message was sent to the associated queue. If a message has
100978 been sent to the corresponding message queue, the hour, minute, and second
100979 of the last time a message was sent to the queue shall be written using the
100980 format `%d:%2.2d:%2.2d`. Otherwise, the format `" no-entry"` shall be
100981 written.

100982 RTIME (**a**,**t**) The time the last message was received from the associated queue. If a
100983 message has been received from the corresponding message queue, the hour,
100984 minute, and second of the last time a message was received from the queue

100985 shall be written using the format `%d:%2.2d:%2.2d`. Otherwise, the format
100986 `" no-entry"` shall be written.

100987 The following eight columns shall be only written out for shared memory segments.

100988 CREATOR (**a**,**c**) The user of the creator of the facility entry. If the user name of the creator is
100989 found in the user database, at least the first eight column positions of the
100990 name shall be written using the format `%s`. Otherwise, the user ID of the
100991 creator shall be written using the format `%d`.

100992 CGROUP (**a**,**c**) The group name of the creator of the facility entry. If the group name of the
100993 creator is found in the group database, at least the first eight column positions
100994 of the name shall be written using the format `%s`. Otherwise, the group ID of
100995 the creator shall be written using the format `%d`.

100996 NATTCH (**a**,**o**) The number of processes attached to the associated shared memory segment.
100997 This field shall be written using the format `%d`.

100998 SEGSZ (**a**,**b**) The size of the associated shared memory segment. This field shall be written
100999 using the format `%d`.

101000 CPID (**a**,**p**) The process ID of the creator of the shared memory entry. This field shall be
101001 written using the format `%d`.

101002 LPID (**a**,**p**) The process ID of the last process to attach or detach the shared memory
101003 segment. This field shall be written using the format:

101004 `"%d", <`*pid*`>`

101005 where <*pid*> is 0 if no process has attached the corresponding shared memory
101006 segment; otherwise, <*pid*> shall be the process ID of the last process to attach
101007 or detach the segment.

101008 ATIME (**a**,**t**) The time the last attach on the associated shared memory segment was
101009 completed. If the corresponding shared memory segment has ever been
101010 attached, the hour, minute, and second of the last time the segment was
101011 attached shall be written using the format `%d:%2.2d:%2.2d`. Otherwise, the
101012 format `" no-entry"` shall be written.

101013 DTIME (**a**,**t**) The time the last detach on the associated shared memory segment was
101014 completed. If the corresponding shared memory segment has ever been
101015 detached, the hour, minute, and second of the last time the segment was
101016 detached shall be written using the format `%d:%2.2d:%2.2d`. Otherwise, the
101017 format `" no-entry"` shall be written.

101018 The following four columns shall be only written out for semaphore sets:

101019 CREATOR (**a**,**c**) The user of the creator of the facility entry. If the user name of the creator is
101020 found in the user database, at least the first eight column positions of the
101021 name shall be written using the format `%s`. Otherwise, the user ID of the
101022 creator shall be written using the format `%d`.

101023 CGROUP (**a**,**c**) The group name of the creator of the facility entry. If the group name of the
101024 creator is found in the group database, at least the first eight column positions
101025 of the name shall be written using the format `%s`. Otherwise, the group ID of
101026 the creator shall be written using the format `%d`.

NSEMS (**a**,**b**) The number of semaphores in the set associated with the semaphore entry. This field shall be written using the format `%d`.

OTIME (**a**,**t**) The time the last semaphore operation on the set associated with the semaphore entry was completed. If a semaphore operation has ever been performed on the corresponding semaphore set, the hour, minute, and second of the last semaphore operation on the semaphore set shall be written using the format `%d:%2.2d:%2.2d`. Otherwise, the format `" no-entry"` shall be written.

The following column shall be written for all three reports when it is requested:

CTIME (**a**,**t**) The time the associated entry was created or changed. The hour, minute, and second of the time when the associated entry was created shall be written using the format `%d:%2.2d:%2.2d`.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Things can change while *ipcs* is running; the information it gives is guaranteed to be accurate only when it was retrieved.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

ipcrm

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *msgrcv*( ), *msgsnd*( ), *semget*( ), *semop*( ), *shmat*( ), *shmdt*( ), *shmget*( )

CHANGE HISTORY

First released in Issue 5.

Issue 6

The Open Group Corrigendum U020/1 is applied, correcting the SYNOPSIS.

The Open Group Corrigenda U032/1 and U032/2 are applied, clarifying the output format.

The Open Group Base Resolution bwg98-004 is applied.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

SD5-XCU-ERN-139 is applied, adding the *ipcrm* utility to the SEE ALSO section.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0108 [584] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

jobs — display status of jobs in the current shell execution environment

SYNOPSIS

UP

```
jobs [-l|-p] [job_id...]
```

DESCRIPTION

If the current shell execution environment (see Section 2.13, on page 2522) is not a subshell environment, the *jobs* utility shall display the status of background jobs that were created in the current shell execution environment; it may also do so if the current shell execution environment is a subshell environment.

When *jobs* reports the termination status of a job, the shell shall remove the job from the background jobs list and the associated process ID from the list of those ``known in the current shell execution environment''; see Section 2.9.3.1 (on page 2506). If a write error occurs when *jobs* writes to standard output, some process IDs might have been removed from the list but not successfully reported.

OPTIONS

The *jobs* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–l (The letter ell.) Provide more information about each job listed. See STDOUT for details.

–p Display only the process IDs for the process group leaders of job-control background jobs and the process IDs associated with non-job-control background jobs (if supported).

By default, the *jobs* utility shall display the status of all background jobs, both running and suspended, and all jobs whose status has changed and have not been reported by the shell.

OPERANDS

The following operand shall be supported:

job_id Specifies the jobs for which the status is to be displayed. If no *job_id* is given, the status information for all jobs shall be displayed. The format of *job_id* is described in XBD Section 3.182 (on page 57).

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *jobs*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If the –**p** option is specified, the output shall consist of one line for each process ID:

`"%d\n", <process ID>`

Otherwise, if the –**l** option is not specified, the output shall be a series of lines of the form:

`"[%d] %c %s %s\n", <job-number>, <current>, <state>, <command>`

where the fields shall be as follows:

<*current*> The character `'+'` identifies the job that would be used as a default for the *fg* or *bg* utilities; this job can also be specified using the *job_id* %+ or `"%%"`. The character `'-'` identifies the job that would become the default if the current default job were to exit; this job can also be specified using the *job_id* %–. For other jobs, this field is a **<**space**>**. At most one job can be identified with `'+'` and at most one job can be identified with `'-'`. If there is any suspended job, then the current job shall be a suspended job. If there are at least two suspended jobs, then the previous job also shall be a suspended job.

<*job-number*> A number that can be used to identify the job to the *wait*, *fg*, *bg*, and *kill* utilities. Using these utilities, the job can be identified by prefixing the job number with `'%'`.

<*state*> One of the following strings (in the POSIX locale):

Running Indicates that the job has not been suspended by a signal and has not exited.

Done Indicates that the job completed and returned exit status zero.

Done(*code*) Indicates that the job completed normally and that it exited with the specified non-zero exit status, *code*, expressed as a decimal number.

Stopped Indicates that the job was suspended by the SIGTSTP signal.

Stopped (**SIGTSTP**)
Indicates that the job was suspended by the SIGTSTP signal.

Stopped (**SIGSTOP**)
Indicates that the job was suspended by the SIGSTOP signal.

Stopped (**SIGTTIN**)
Indicates that the job was suspended by the SIGTTIN signal.

Stopped (**SIGTTOU**)
Indicates that the job was suspended by the SIGTTOU signal.

The implementation may substitute the string **Suspended** in place of **Stopped**. If the job was terminated by a signal, the format of **<***state***>** is unspecified, but it shall be visibly distinct from all of the other **<***state***>** formats shown here and shall

indicate the name or description of the signal causing the termination.

<*command*> The associated command that was given to the shell.

If the **–l** option is specified:

- For job-control background jobs, a field containing the process group ID shall be inserted before the **<*state*>** field. Also, more processes in a process group may be output on separate lines, using only the process ID and **<*command*>** fields.
- For non-job-control background jobs (if supported), a field containing the process ID associated with the job shall be inserted before the **<*state*>** field. Also, more processes created to execute the job may be output on separate lines, using only the process ID and **<*command*>** fields.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 The output specified in STDOUT was successfully written to standard output.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

This utility is required to be intrinsic. See Section 1.7 (on page 2470) for details.

The **–p** option is the only portable way to find out the process group of a job-control background job because different implementations have different strategies for defining the process group of the job. Usage such as $(*jobs* –**p**) provides a way of referring to the process group of the job in an implementation-independent way.

The *jobs* utility does not work as expected when it is operating in its own utility execution environment because that environment has no applicable jobs to manipulate. See the APPLICATION USAGE section for *bg*. For this reason, *jobs* is generally implemented as a shell regular built-in.

EXAMPLES

None.

RATIONALE

Both `"%%"` and `"%+"` are used to refer to the current job. Both forms are of equal validity—the `"%%"` mirroring `"$$"` and `"%+"` mirroring the output of *jobs*. Both forms reflect historical practice of the KornShell and the C shell with job control.

The job control features provided by *bg*, *fg*, and *jobs* are based on the KornShell. The standard developers examined the characteristics of the C shell versions of these utilities and found that differences exist. Despite widespread use of the C shell, the KornShell versions were selected for this volume of POSIX.1-2024 to maintain a degree of uniformity with the rest of the KornShell features selected (such as the very popular command line editing features).

The *jobs* utility is not dependent on job control being enabled, as are the seemingly related *bg* and *fg* utilities because *jobs* is useful for examining background jobs, regardless of the current state of job control. When job control has been disabled using *set* **+m**, the *jobs* utility can still be used to examine the job-control background jobs and (if supported) non-job-control background jobs that were created in the current shell execution environment. See also the RATIONALE for *kill* and *wait*.

The output for terminated jobs is left unspecified to accommodate various historical systems. The following formats have been witnessed:

1. **Killed**(*signal name*)
2. *signal name*
3. *signal name*(**coredump**)
4. *signal description*– **core dumped**

Most users should be able to understand these formats, although it means that applications have trouble parsing them.

The calculation of job IDs was not described since this would suggest an implementation, which may impose unnecessary restrictions.

In an early proposal, a –**n** option was included to ``Display the status of jobs that have changed, exited, or stopped since the last status report''. It was removed because the shell always writes any changed status of jobs before each prompt.

If *jobs* uses buffered writes to standard output, a write error could be detected when attempting to flush a buffer containing multiple reports of terminated jobs, resulting in some unreported jobs having their process IDs removed from the list of those known in the current shell execution environment (because they were removed when the report was added to the buffer).

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.13 (on page 2522), *bg*, *fg*, *kill*, *wait*

XBD Section 3.182 (on page 57), Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The JC shading is removed as job control is mandatory in this version.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that this utility is required to be intrinsic.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1254 is applied, updating various requirements for the *jobs* utility to account for the addition of Section 2.11 (on page 2518).

Austin Group Defect 1492 is applied, clarifying the requirements when a write error to standard

101253 output occurs.

NAME

join — relational database operator

SYNOPSIS

```
join [-a file_number|-v file_number] [-e string] [-o list] [-t char]
    [-1 field] [-2 field] file1 file2
```

DESCRIPTION

The *join* utility shall perform an equality join on the files *file1* and *file2*. The joined files shall be written to the standard output.

The join field is a field in each file on which the files are compared. The *join* utility shall write one line in the output for each pair of lines in *file1* and *file2* that have join fields that collate equally. The output line by default shall consist of the join field, then the remaining fields from *file1*, then the remaining fields from *file2*. This format can be changed by using the **−o** option (see below). The **−a** option can be used to add unmatched lines to the output. The **−v** option can be used to output only unmatched lines.

The files *file1* and *file2* shall be ordered in the collating sequence of *sort* **−b** on the fields on which they shall be joined, by default the first in each line. All selected output shall be written in the same collating sequence.

The default input field separators shall be <blank> characters. In this case, multiple separators shall count as one field separator, and leading separators shall be ignored. The default output field separator shall be a <space>.

The field separator and collating sequence can be changed by using the **−t** option (see below).

If the same key appears more than once in either file, all combinations of the set of remaining fields in *file1* and the set of remaining fields in *file2* are output in the order of the lines encountered.

If the input files are not in the appropriate collating sequence, the results are unspecified.

OPTIONS

The *join* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−a *file_number*
: Produce a line for each unpairable line in file *file_number*, where *file_number* is 1 or 2, in addition to the default output. If both **−a**1 and **−a**2 are specified, all unpairable lines shall be output.

−e *string*
: Replace empty output fields in the list selected by **−o** with the string *string*.

−o *list*
: Construct the output line to comprise the fields specified in *list*, each element of which shall have one of the following two forms:

  1. *file_number.field*, where *file_number* is a file number and *field* is a decimal integer field number
  2. 0 (zero), representing the join field

  The elements of *list* shall be either <comma>-separated or <blank>-separated, as specified in Guideline 8 of XBD Section 12.2 (on page 215). The fields specified by *list* shall be written for all selected output lines. Fields selected by *list* that do not appear in the input shall be treated as empty output fields. (See the **−e** option.) Only specifically requested fields shall be written. The application shall ensure that *list* is a single command line argument.

101298 **–t** *char* Use character *char* as a separator, for both input and output. Every appearance of
101299 *char* in a line shall be significant. When this option is specified, the collating
101300 sequence shall be the same as *sort* without the **–b** option.

101301 **–v** *file_number*
101302 Instead of the default output, produce a line only for each unpairable line in
101303 *file_number*, where *file_number* is 1 or 2. If both **–v**1 and **–v**2 are specified, all
101304 unpairable lines shall be output.

101305 **–1** *field* Join on the *field*th field of file 1. Fields are decimal integers starting with 1.

101306 **–2** *field* Join on the *field*th field of file 2. Fields are decimal integers starting with 1.

101307 **OPERANDS**

101308 The following operands shall be supported:

101309 *file1*, *file2* A pathname of a file to be joined. If either of the *file1* or *file2* operands is `'–'`, the
101310 standard input shall be used in its place.

101311 **STDIN**

101312 The standard input shall be used only if the *file1* or *file2* operand is `'–'`. See the INPUT FILES
101313 section.

101314 **INPUT FILES**

101315 The input files shall be text files.

101316 **ENVIRONMENT VARIABLES**

101317 The following environment variables shall affect the execution of *join*:

101318 *LANG* Provide a default value for the internationalization variables that are unset or null.
101319 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
101320 variables used to determine the values of locale categories.)

101321 *LC_ALL* If set to a non-empty string value, override the values of all the other
101322 internationalization variables.

101323 *LC_COLLATE*
101324 Determine the locale of the collating sequence *join* expects to have been used when
101325 the input files were sorted.

101326 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
101327 characters (for example, single-byte as opposed to multi-byte characters in
101328 arguments and input files).

101329 *LC_MESSAGES*
101330 Determine the locale that should be used to affect the format and contents of
101331 diagnostic messages written to standard error.

101332 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

101333 **ASYNCHRONOUS EVENTS**

101334 Default.

101335 **STDOUT**

101336 The *join* utility output shall be a concatenation of selected character fields. When the **–o** option
101337 is not specified, the output shall be:

101338 `"%s%s%s\n", <join field>, <other file1 fields>,`
101339 `<other file2 fields>`

101340 If the join field is not the first field in a file, the **<***other file fields***>** for that file shall be:

101341 `<fields preceding join field>, <fields following join field>`

101342 When the **–o** option is specified, the output format shall be:

101343 `"%s\n", <concatenation of fields>`

101344 where the concatenation of fields is described by the **–o** option, above.

101345 For either format, each field (except the last) shall be written with its trailing separator character.
101346 If the separator is the default (**<**blank**>** characters), a single **<**space**>** shall be written after each
101347 field (except the last).

101348 **STDERR**

101349 The standard error shall be used only for diagnostic messages.

101350 **OUTPUT FILES**

101351 None.

101352 **EXTENDED DESCRIPTION**

101353 None.

101354 **EXIT STATUS**

101355 The following exit values shall be returned:

101356 0 All input files were output successfully.

101357 **>**0 An error occurred.

101358 **CONSEQUENCES OF ERRORS**

101359 Default.

101360 **APPLICATION USAGE**

101361 Pathnames consisting of numeric digits or of the form *string.string* should not be specified
101362 directly following the **–o** list.

101363 If the collating sequence of the current locale does not have a total ordering of all characters (see
101364 XBD Section 7.3.2, on page 139), *join* treats fields that collate equally but are not identical as
101365 being the same. If this behavior is not desired, it can be avoided by forcing the use of the POSIX
101366 locale (although this means re-sorting the input files into the POSIX locale collating sequence.)

101367 When using *join* to process pathnames, it is recommended that LC_ALL, or at least LC_CTYPE
101368 and LC_COLLATE, are set to POSIX or C in the environment, since pathnames can contain byte
101369 sequences that do not form valid characters in some locales, in which case the utility's behavior
101370 would be undefined. In the POSIX locale each byte is a valid single-byte character, and therefore
101371 this problem is avoided.

101372 **EXAMPLES**

101373 The **–o** 0 field essentially selects the union of the join fields. For example, given file **phone**:

```
101374 !Name           Phone Number
101375 Don             +1 123-456-7890
101376 Hal             +1 234-567-8901
101377 Yasushi         +2 345-678-9012
```

101378 and file **fax**:

```
101379 !Name           Fax Number
101380 Don             +1 123-456-7899
101381 Keith           +1 456-789-0122
101382 Yasushi         +2 345-678-9011
```


(where the large expanses of white space are meant to each represent a single <tab>), the command:

```
join -t "<tab>" -a 1 -a 2 -e '(unknown)' -o 0,1.2,2.2 phone fax
```

(where <tab> is a literal <tab> character) would produce:

```
!Name           Phone Number            Fax Number
Don             +1 123-456-7890         +1 123-456-7899
Hal             +1 234-567-8901         (unknown)
Keith           (unknown)               +1 456-789-0122
Yasushi         +2 345-678-9012         +2 345-678-9011
```

Multiple instances of the same key will produce combinatorial results. The following:

```
fa:
    a x
    a y
    a z
fb:
    a p
```

will produce:

```
a x p
a y p
a z p
```

And the following:

```
fa:
    a b c
    a d e
fb:
    a w x
    a y z
    a o p
```

will produce:

```
a b c w x
a b c y z
a b c o p
a d e w x
a d e y z
a d e o p
```

RATIONALE

The **-e** option is only effective when used with **-o** because, unless specific fields are identified using **-o**, *join* is not aware of what fields might be empty. The exception to this is the join field, but identifying an empty join field with the **-e** string is not historical practice and some scripts might break if this were changed.

The 0 field in the **-o** list was adopted from the Tenth Edition version of *join* to satisfy international objections that the *join* in the base documents for IEEE Std 1003.2-1992 did not support the ``full join'' or ``outer join'' described in relational database literature. Although it has been possible to include a join field in the output (by default, or by field number using **-o**), the join field could not be included for an unpaired line selected by **-a**. The **-o** 0 field essentially

selects the union of the join fields.

This sort of outer join was not possible with the *join* commands in the base documents for IEEE Std 1003.2-1992. The **–o** 0 field was chosen because it is an upwards-compatible change for applications. An alternative was considered: have the join field represent the union of the fields in the files (where they are identical for matched lines, and one or both are null for unmatched lines). This was not adopted because it would break some historical applications.

The ability to specify *file2* as – is not historical practice; it was added for completeness.

The **–v** option is not historical practice, but was considered necessary because it permitted the writing of *only* those lines that do not match on the join field, as opposed to the **–a** option, which prints both lines that do and do not match. This additional facility is parallel with the **–v** option of *grep*.

Some historical implementations have been encountered where a blank line in one of the input files was considered to be the end of the file; the description in this volume of POSIX.1-2024 does not cite this as an allowable case.

Earlier versions of this standard allowed **–j**, **–j1**, **–j2** options, and a form of the **–o** option that allowed the *list* option-argument to be multiple arguments. These forms are no longer specified by POSIX.1-2024 but may be present in some implementations.

FUTURE DIRECTIONS

None.

SEE ALSO

awk, *comm*, *sort*, *uniq*

XBD Section 7.3.2 (on page 139), Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The obsolescent **–j** options and the multi-argument **–o** option are removed in this version.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #027 is applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0109 [963], XCU/TC2-2008/0110 [663], XCU/TC2-2008/0111 [971], and XCU/TC2-2008/0112 [885] are applied.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

kill — terminate or signal processes

SYNOPSIS

```
kill [-s signal_name] pid...
kill -l [exit_status]
```

XSI

```
kill [-signal_name] pid...
kill [-signal_number] pid...
```

DESCRIPTION

The *kill* utility shall send a signal to the process or processes specified by each *pid* operand.

For each *pid* operand, the *kill* utility shall perform actions equivalent to the *kill*() function defined in the System Interfaces volume of POSIX.1-2024 called with the following arguments:

- The value of the *pid* operand shall be used as the *pid* argument.
- The *sig* argument is the value specified by the **–s** option, *–signal_number* option, or the *–signal_name* option, or by SIGTERM, if none of these options is specified.

OPTIONS

XSI The *kill* utility shall conform to XBD Section 12.2 (on page 215), except that in the last two SYNOPSIS forms, the *–signal_number* and *–signal_name* options are usually more than a single character.

The following options shall be supported:

–l (The letter ell.) Write all values of *signal_name* supported by the implementation, if no operand is given. If an *exit_status* operand is given and it is a value of the `'?'` shell special parameter (see Section 2.5.2 (on page 2479) and *wait*) corresponding to a process that was terminated or stopped by a signal, the *signal_name* corresponding to the signal that terminated or stopped the process shall be written. If an *exit_status* operand is given and it is the unsigned decimal integer value of a signal number, the *signal_name* (the symbolic constant name without the **SIG** prefix defined in the Base Definitions volume of POSIX.1-2024) corresponding to that signal shall be written. Otherwise, the results are unspecified.

–s *signal_name*
Specify the signal to send, using one of the symbolic names defined in the **<signal.h>** header. Values of *signal_name* shall be recognized in a case-independent fashion, without the **SIG** prefix. In addition, the symbolic name 0 shall be recognized, representing the signal value zero. The corresponding signal shall be sent instead of SIGTERM.

XSI *–signal_name*
Equivalent to **–s** *signal_name*.

XSI *–signal_number*
Specify a non-negative decimal integer, *signal_number*, representing the signal to be used instead of SIGTERM, as the *sig* argument in the effective call to *kill*(). The correspondence between integer values and the *sig* value used is shown in the following list.

The effects of specifying any *signal_number* other than those listed below are undefined.

0	0
1	SIGHUP
2	SIGINT
3	SIGQUIT
6	SIGABRT
9	SIGKILL
14	SIGALRM
15	SIGTERM

If the first argument is a negative integer, it shall be interpreted as a *–signal_number* option, not as a negative *pid* operand specifying a process group.

OPERANDS

The following operands shall be supported:

pid One of the following:

1. A decimal integer specifying a process or process group to be signaled. The process or processes selected by positive, negative, and zero values of the *pid* operand shall be as described for the *kill*( ) function. If process number 0 is specified, all processes in the current process group shall be signaled. For the effects of negative *pid* numbers, see the *kill*( ) function defined in the System Interfaces volume of POSIX.1-2024. If the first *pid* operand is negative, it should be preceded by `"--"` to keep it from being interpreted as an option.

2. A job ID (see XBD Section 3.182, on page 57) that identifies a process group in the case of a job-control background job, or a process ID in the case of a non-job-control background job (if supported), to be signaled. The job ID notation is applicable only for invocations of *kill* in the current shell execution environment; see Section 2.13 (on page 2522).

   Note: The job ID type of *pid* is only available on systems supporting the User Portability Utilities option or supporting non-job-control background jobs.

exit_status A decimal integer specifying a signal number or the exit status of a process terminated by a signal.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *kill*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

When the **–l** option is not specified, the standard output shall not be used.

When the **–l** option is specified, the symbolic name of each signal shall be written in the following format:

```
"%s%c", <signal_name>, <separator>
```

where the <*signal_name*> is in uppercase, without the **SIG** prefix, and the <*separator*> shall be either a <newline> or a <space>. For the last signal written, <*separator*> shall be a <newline>.

When both the **–l** option and *exit_status* operand are specified, the symbolic name of the corresponding signal shall be written in the following format:

```
"%s\n", <signal_name>
```

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 The **–l** option was specified and the output specified in STDOUT was successfully written to standard output; or, the **–l** option was not specified, at least one matching process was found for each *pid* operand, and the specified signal was successfully processed for at least one matching process.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

This utility is required to be intrinsic. See Section 1.7 (on page 2470) for details.

Process numbers can be found by using *ps*.

The use of job ID notation is not dependent on job control being enabled. When job control has been disabled using *set* **+m**, *kill* can still be used to signal the process group associated with a job-control background job, or the process ID associated with a non-control background job (if supported), using

```
kill %<background job number>
```

See also the RATIONALE for *jobs* and *wait*.

The job ID notation is not required to work as expected when *kill* is operating in its own utility execution environment. In either of the following examples:

```
nohup kill %1 &
system("kill %1");
```

the *kill* operates in a different environment and does not share the shell's understanding of job numbers.

EXAMPLES

Any of the commands:

```
kill -9 100 -165
kill -s kill 100 -165
kill -s KILL 100 -165
```

sends the SIGKILL signal to the process whose process ID is 100 and to all processes whose process group ID is 165, assuming the sending process has permission to send that signal to the specified processes, and that they exist.

The System Interfaces volume of POSIX.1-2024 and this volume of POSIX.1-2024 do not require specific signal numbers for any *signal_names*. Even the *–signal_number* option provides symbolic (although numeric) names for signals. If a process is terminated by a signal, its exit status indicates the signal that killed it, but the exact values are not specified. The *kill* **–l** option, however, can be used to map decimal signal numbers and exit status values into the name of a signal. The following example reports the status of a terminated job:

```
job
stat=$?
if [ $stat -eq 0 ]
then
    echo job completed successfully.
elif [ $stat -gt 128 ]
then
    echo job terminated by signal SIG$(kill -l $stat).
else
    echo job terminated with error code $stat.
fi
```

To send the default signal to a process group (say 123), an application should use a command similar to one of the following:

```
kill -s TERM -- -123
kill -- -123
```


RATIONALE

The **–l** option originated from the C shell, and is also implemented in the KornShell. The C shell output can consist of multiple output lines because the signal names do not always fit on a single line on some terminal screens. The KornShell output also included the implementation-defined signal numbers and was considered by the standard developers to be too difficult for scripts to parse conveniently. The specified output format is intended not only to accommodate the historical C shell output, but also to permit an entirely vertical or entirely horizontal listing on systems for which this is appropriate.

An early proposal invented the name SIGNULL as a *signal_name* for signal 0 (used by the System Interfaces volume of POSIX.1-2024 to test for the existence of a process without sending it a signal). Since the *signal_name* 0 can be used in this case unambiguously, SIGNULL has been removed.

An early proposal also required symbolic *signal_name*s to be recognized with or without the **SIG** prefix. Historical versions of *kill* have not written the **SIG** prefix for the **–l** option and have not recognized the **SIG** prefix on *signal_name*s. Since neither applications portability nor ease-of-use would be improved by requiring this extension, it is no longer required.

To avoid an ambiguity of an initial negative number argument specifying either a signal number or a process group, POSIX.1-2024 mandates that it is always considered the former by implementations that support the XSI option. It also requires that conforming applications always use the `"--"` options terminator argument when specifying a process group.

The **–s** option was added in response to international interest in providing some form of *kill* that meets the Utility Syntax Guidelines.

The job ID notation is not required to work as expected when *kill* is operating in its own utility execution environment. In either of the following examples:

```
nohup kill %1 &
system("kill %1");
```

the *kill* operates in a different environment and does not understand how the shell has managed its job numbers.

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 2 (on page 2472), *ps*, *wait*

XBD Section 3.182 (on page 57), Chapter 8 (on page 167), Section 12.2 (on page 215), **<signal.h>**

XSH *kill*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

The obsolescent versions of the SYNOPSIS are turned into non-obsolescent features of the XSI option, corresponding to a similar change in the *trap* special built-in.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that this utility is required to be intrinsic.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1254 is applied, clarifying the **–l** option with regard to an *exit_status* operand corresponding to a stopped process, changing ``job control job ID'' to ``job ID'', and adding a paragraph to the RATIONALE section.

Austin Group Defect 1260 is applied, changing the SYNOPSIS and EXAMPLES sections in relation to the **–s** option, and the RATIONALE section in relation to the use of **"--"** when specifying a process group.

Austin Group Defect 1504 is applied, changing the EXIT STATUS section.

NAME

lex — generate programs for lexical tasks (**DEVELOPMENT**)

SYNOPSIS

CD

```
lex [-t] [-n|-v] [file...]
```

DESCRIPTION

The *lex* utility shall generate C programs to be used in lexical processing of character input, and that can be used as an interface to *yacc*. The C programs shall be generated from *lex* source code and conform to the ISO C standard, without depending on any undefined, unspecified, or implementation-defined behavior, except in cases where the code is copied directly from the supplied source, or in cases that are documented by the implementation. Usually, the *lex* utility shall write the program it generates to the file **lex.yy.c**; the state of this file is unspecified if *lex* exits with a non-zero exit status. See the EXTENDED DESCRIPTION section for a complete description of the *lex* input language.

OPTIONS

The *lex* utility shall conform to XBD Section 12.2 (on page 215), except for Guideline 9.

The following options shall be supported:

−n Suppress the summary of statistics usually written with the **−v** option. If no table sizes are specified in the *lex* source code and the **−v** option is not specified, then **−n** is implied.

−t Write the resulting program to standard output instead of **lex.yy.c**.

−v Write a summary of *lex* statistics to the standard output. (See the discussion of *lex* table sizes in Definitions in lex (on page 3039).) If the **−t** option is specified and **−n** is not specified, this report shall be written to standard error. If table sizes are specified in the *lex* source code, and if the **−n** option is not specified, the **−v** option may be enabled.

OPERANDS

The following operand shall be supported:

file A pathname of an input file. If more than one such *file* is specified, all files shall be concatenated to produce a single *lex* program. If no *file* operands are specified, or if a *file* operand is `'−'`, the standard input shall be used.

STDIN

The standard input shall be used if no *file* operands are specified, or if a *file* operand is `'−'`. See INPUT FILES.

INPUT FILES

The input files shall be text files containing *lex* source code, as described in the EXTENDED DESCRIPTION section.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *lex*:

LANG Provide a default value for the internationalization variables that are unset or null.

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.

101721 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
101722 characters (for example, single-byte as opposed to multi-byte characters in
101723 arguments and input files), and the behavior of character classes within regular
101724 expressions.

101725 *LC_MESSAGES*
101726 Determine the locale that should be used to affect the format and contents of
101727 diagnostic messages written to standard error.

101728 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

101729 If the values (if any) of the *LANG*, *LC_COLLATE*, *LC_CTYPE*, and *LC_ALL* variables result in the
101730 locale in effect for the *LC_CTYPE* or *LC_COLLATE* category not being the POSIX locale, the
101731 behavior is unspecified. (See XBD Section 8.2 (on page 169) for the precedence of
101732 internationalization variables used to determine the values of locale categories.)

101733 **ASYNCHRONOUS EVENTS**
101734 Default.

101735 **STDOUT**
101736 If the **–t** option is specified, the text file of C source code output of *lex* shall be written to
101737 standard output.

101738 If the **–t** option is not specified:

101739 • Implementation-defined informational, error, and warning messages concerning the
101740 contents of *lex* source code input shall be written to either the standard output or standard
101741 error.

101742 • If the **–v** option is specified and the **–n** option is not specified, *lex* statistics shall also be
101743 written to either the standard output or standard error, in an implementation-defined
101744 format. These statistics may also be generated if table sizes are specified with a `'%'`
101745 operator in the *Definitions* section, as long as the **–n** option is not specified.

101746 **STDERR**
101747 If the **–t** option is specified, implementation-defined informational, error, and warning messages
101748 concerning the contents of *lex* source code input shall be written to the standard error.

101749 If the **–t** option is not specified:

101750 1. Implementation-defined informational, error, and warning messages concerning the
101751 contents of *lex* source code input shall be written to either the standard output or
101752 standard error.

101753 2. If the **–v** option is specified and the **–n** option is not specified, *lex* statistics shall also be
101754 written to either the standard output or standard error, in an implementation-defined
101755 format. These statistics may also be generated if table sizes are specified with a `'%'`
101756 operator in the *Definitions* section, as long as the **–n** option is not specified.

101757 **OUTPUT FILES**
101758 A text file containing C source code shall be written to **lex.yy.c**, or to the standard output if the **–t**
101759 option is present.

101760 **EXTENDED DESCRIPTION**
101761 Each input file shall contain *lex* source code, which is a table of regular expressions with
101762 corresponding actions in the form of C program fragments.

101763 When **lex.yy.c** is compiled and linked with the *lex* library (using the **–l l** operand with *c17*), the
101764 resulting program shall read character input from the standard input and shall partition it into

strings that match the given expressions.

When an expression is matched, these actions shall occur:

- The input string that was matched shall be left in *yytext* as a null-terminated string; *yytext* shall either be an external character array or a pointer to a character string. As explained in Definitions in lex, the type can be explicitly selected using the **%array** or **%pointer** declarations, but the default is implementation-defined.
- The external **int** *yyleng* shall be set to the length of the matching string.
- The expression's corresponding program fragment, or action, shall be executed.

During pattern matching, *lex* shall search the set of patterns for the single longest possible match. Among rules that match the same number of characters, the rule given first shall be chosen.

The general format of *lex* source shall be:

Definitions
%%
Rules
%%
*User*Subroutines

The first `"%%"` is required to mark the beginning of the rules (regular expressions and actions); the second `"%%"` is required only if user subroutines follow.

Any line in the *Definitions* section beginning with a <blank> shall be assumed to be a C program fragment and shall be copied to the external definition area of the **lex.yy.c** file. Similarly, anything in the *Definitions* section included between delimiter lines containing only `"%{"` and `"%}"` shall also be copied unchanged to the external definition area of the **lex.yy.c** file.

Any such input (beginning with a <blank> or within `"%{"` and `"%}"` delimiter lines) appearing at the beginning of the *Rules* section before any rules are specified shall be written to **lex.yy.c** after the declarations of variables for the *yylex*() function and before the first line of code in *yylex*(). Thus, user variables local to *yylex*() can be declared here, as well as application code to execute upon entry to *yylex*().

The action taken by *lex* when encountering any input beginning with a <blank> or within `"%{"` and `"%}"` delimiter lines appearing in the *Rules* section but coming after one or more rules is undefined. The presence of such input may result in an erroneous definition of the *yylex*() function.

C-language code in the input shall not contain C-language trigraphs. The C-language code within `"%{"` and `"%}"` delimiter lines shall not contain any lines consisting only of `"%}"`, or only of `"%%"`.

Definitions in lex

Definitions appear before the first `"%%"` delimiter. Any line in this section not contained between `"%{"` and `"%}"` lines and not beginning with a <blank> shall be assumed to define a *lex* substitution string. The format of these lines shall be:

```
name substitute
```

If a *name* does not meet the requirements for identifiers in the ISO C standard, the result is undefined. The string *substitute* shall replace the string {*name*} when it is used in a rule. The *name* string shall be recognized in this context only when the braces are provided and when it does

not appear within a bracket expression or within double-quotes.

In the *Definitions* section, any line beginning with a <percent-sign> (`'%'`) character and followed by an alphanumeric word beginning with either `'s'` or `'S'` shall define a set of start conditions. Any line beginning with a `'%'` followed by a word beginning with either `'x'` or `'X'` shall define a set of exclusive start conditions. When the generated scanner is in a `%s` state, patterns with no state specified shall be also active; in a `%x` state, such patterns shall not be active. The rest of the line, after the first word, shall be considered to be one or more <blank>-separated names of start conditions. Start condition names shall be constructed in the same way as definition names. Start conditions can be used to restrict the matching of regular expressions to one or more states as described in Regular Expressions in lex (on page 3041).

Implementations shall accept either of the following two mutually-exclusive declarations in the *Definitions* section:

%array Declare the type of *yytext* to be a null-terminated character array.

%pointer Declare the type of *yytext* to be a pointer to a null-terminated character string.

The default type of *yytext* is implementation-defined. If an application refers to *yytext* outside of the scanner source file (that is, via an **extern**), the application shall include the appropriate **%array** or **%pointer** declaration in the scanner source file.

Implementations shall accept declarations in the *Definitions* section for setting certain internal table sizes. The declarations are shown in the following table.

Table 3-11 Table Size Declarations in *lex*

Declaration	Description	Minimum Value
%**p** *n*	Number of positions	2 500
%**n** *n*	Number of states	500
%**a** *n*	Number of transitions	2 000
%**e** *n*	Number of parse tree nodes	1 000
%**k** *n*	Number of packed character classes	1 000
%**o** *n*	Size of the output array	3 000

In the table, *n* represents a positive decimal integer, preceded by one or more <blank> characters. The exact meaning of these table size numbers is implementation-defined. The implementation shall document how these numbers affect the *lex* utility and how they are related to any output that may be generated by the implementation should limitations be encountered during the execution of *lex*. It shall be possible to determine from this output which of the table size values needs to be modified to permit *lex* to successfully generate tables for the input language. The values in the column Minimum Value represent the lowest values conforming implementations shall provide.

Rules in lex

The rules in *lex* source files are a table in which the left column contains regular expressions and the right column contains actions (C program fragments) to be executed when the expressions are recognized.

```
ERE action
ERE action
...
```

The extended regular expression (ERE) portion of a row shall be separated from *action* by one or more <blank> characters. A regular expression containing <blank> characters shall be

recognized under one of the following conditions:

- The entire expression appears within double-quotes.
- The <blank> characters appear within double-quotes or square brackets.
- Each <blank> is preceded by a <backslash> character.

User Subroutines in lex

Anything in the user subroutines section shall be copied to **lex.yy.c** following *yylex*().

Regular Expressions in lex

The *lex* utility shall support the set of extended regular expressions (see XBD Section 9.4, on page 187), with the following additions and exceptions to the syntax:

`"..."` Any string enclosed in double-quotes shall represent the characters within the double-quotes as themselves, except that <backslash>-escapes (which appear in the following table) shall be recognized. Any <backslash>-escape sequence shall be terminated by the closing quote. For example, `"\01""1"` represents a single string: the octal value 1 followed by the character `'1'`.

<*state*>*r*, <*state1*,*state2*,...>*r*
The regular expression *r* shall be matched only when the program is in one of the start conditions indicated by *state*, *state1*, and so on; see Actions in lex (on page 3043). (As an exception to the typographical conventions of the rest of this volume of POSIX.1-2024, in this case <*state*> does not represent a metavariable, but the literal angle-bracket characters surrounding a symbol.) The start condition shall be recognized as such only at the beginning of a regular expression.

r/*x* The regular expression *r* shall be matched only if it is followed by an occurrence of regular expression *x* (*x* is the instance of trailing context, further defined below). The token returned in *yytext* shall only match *r*. If the trailing portion of *r* matches the beginning of *x*, the result is unspecified. The *r* expression cannot include further trailing context or the `'$'` (match-end-of-line) operator; *x* cannot include the `'^'` (match-beginning-of-line) operator, nor trailing context, nor the `'$'` operator. That is, only one occurrence of trailing context is allowed in a *lex* regular expression, and the `'^'` operator only can be used at the beginning of such an expression.

{*name*} When *name* is one of the substitution symbols from the *Definitions* section, the string, including the enclosing braces, shall be replaced by the *substitute* value. The *substitute* value shall be treated in the extended regular expression as if it were enclosed in parentheses. No substitution shall occur if {*name*} occurs within a bracket expression or within double-quotes.

Within an ERE, a <backslash> character shall be considered to begin an escape sequence as specified in the table in XBD Chapter 5 (on page 113) (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`). In addition, the escape sequences in the following table shall be recognized.

A literal <newline> cannot occur within an ERE; the escape sequence `'\n'` can be used to represent a <newline>. A <newline> shall not be matched by a period operator.

Table 3-12 Escape Sequences in *lex*

Escape Sequence	Description	Meaning
`\`*digits*	A <backslash> character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the one, two, or three-digit octal integer. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading <backslash> for each byte.
`\x`*digits*	A <backslash> character followed by the longest sequence of hexadecimal-digit characters (01234567abcdefABCDEF). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the hexadecimal integer.
`\c`	A <backslash> character followed by any character not described in this table or in the table in XBD Chapter 5 (on page 113) (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`).	The character `'c'`, unchanged.

Note: If a `'\x'` sequence needs to be immediately followed by a hexadecimal digit character, a sequence such as `"\x1""1"` can be used, which represents a character containing the value 1, followed by the character `'1'`.

The order of precedence given to extended regular expressions for *lex* differs from that specified in XBD Section 9.4 (on page 187). The order of precedence for *lex* shall be as shown in the following table, from high to low.

Note: The escaped characters entry is not meant to imply that these are operators, but they are included in the table to show their relationships to the true operators. The start condition, trailing context, and anchoring notations have been omitted from the table because of the placement restrictions described in this section; they can only appear at the beginning or ending of an ERE.

Table 3-13 ERE Precedence in *lex*

Extended Regular Expression	Precedence
collation-related bracket symbols	`[= =]  [: :]  [. .]`
escaped characters	`\<`*special character*`>`
bracket expression	`[ ]`
quoting	`"..."`
grouping	`( )`
definition	`{`*name*`}`
single-character RE duplication	`* + ?`
concatenation	
interval expression	`{m,n}`
alternation	|

The ERE anchoring operators `'^'` and `'$'` do not appear in the table. With *lex* regular

expressions, these operators are restricted in their use: the `'^'` operator can only be used at the beginning of an entire regular expression, and the `'$'` operator only at the end. The operators apply to the entire regular expression. Thus, for example, the pattern `"(^abc)|(def$)"` is undefined; it can instead be written as two separate rules, one with the regular expression `"^abc"` and one with `"def$"`, which share a common action via the special `'|'` action (see below). If the pattern were written `"^abc|def$"`, it would match either `"abc"` or `"def"` on a line by itself.

Unlike the general ERE rules, embedded anchoring is not allowed by most historical *lex* implementations. An example of embedded anchoring would be for patterns such as `"(^| )foo( |$)"` to match `"foo"` when it exists as a complete word. This functionality can be obtained using existing *lex* features:

```
^foo/[ \n]       |
" foo"/[ \n]    /* Found foo as a separate word. */
```

Note also that `'$'` is a form of trailing context (it is equivalent to `"/\n"`) and as such cannot be used with regular expressions containing another instance of the operator (see the preceding discussion of trailing context).

The additional regular expressions trailing-context operator `'/'` can be used as an ordinary character if presented within double-quotes, `"/"`; preceded by a <backslash>, `"\/"`; or within a bracket expression, `"[/]"`. The start-condition `'<'` and `'>'` operators shall be special only in a start condition at the beginning of a regular expression; elsewhere in the regular expression they shall be treated as ordinary characters.

Actions in lex

The action to be taken when an ERE is matched can be a C program fragment or the special actions described below; the program fragment can contain one or more C statements, and can also include special actions. The empty C statement `';'` shall be a valid action; any string in the **lex.yy.c** input that matches the pattern portion of such a rule is effectively ignored or skipped. However, the absence of an action shall not be valid, and the action *lex* takes in such a condition is undefined.

The specification for an action, including C statements and special actions, can extend across several lines if enclosed in braces:

```
ERE <one or more blanks> { program statement
                           program statement }
```

The program statements shall not contain unbalanced curly brace preprocessing tokens.

The default action when a string in the input to a **lex.yy.c** program is not matched by any expression shall be to copy the string to the output. Because the default behavior of a program generated by *lex* is to read the input and copy it to the output, a minimal *lex* source program that has just `"%%"` shall generate a C program that simply copies the input to the output unchanged.

Four special actions shall be available:

```
|   ECHO;   REJECT;   BEGIN
```

| The action `'|'` means that the action for the next rule is the action for this rule. Unlike the other three actions, `'|'` cannot be enclosed in braces or be <semicolon>-terminated; the application shall ensure that it is specified alone, with no other actions.

ECHO; Write the contents of the string *yytext* on the output.

REJECT; Usually only a single expression is matched by a given string in the input. **REJECT** means ``continue to the next expression that matches the current input'', and shall cause whatever rule was the second choice after the current rule to be executed for the same input. Thus, multiple rules can be matched and executed for one input string or overlapping input strings. For example, given the regular expressions `"xyz"` and `"xy"` and the input `"xyz"`, usually only the regular expression `"xyz"` would match. The next attempted match would start after **z.** If the last action in the `"xyz"` rule is **REJECT**, both this rule and the `"xy"` rule would be executed. The **REJECT** action may be implemented in such a fashion that flow of control does not continue after it, as if it were equivalent to a **goto** to another part of *yylex*( ). The use of **REJECT** may result in somewhat larger and slower scanners.

BEGIN The action:

```
BEGIN newstate;
```

switches the state (start condition) to *newstate*. If the string *newstate* has not been declared previously as a start condition in the *Definitions* section, the results are unspecified. The initial state is indicated by the digit `'0'` or the token **INITIAL**.

The functions or macros described below are accessible to user code included in the *lex* input. It is unspecified whether they appear in the C code output of *lex*, or are accessible only through the **–l l** operand to *c17* (the *lex* library).

int *yylex*(**void**)
Performs lexical analysis on the input; this is the primary function generated by the *lex* utility. The function shall return zero when the end of input is reached; otherwise, it shall return non-zero values (tokens) determined by the actions that are selected.

int *yymore*(**void**)
When called, indicates that when the next input string is recognized, it is to be appended to the current value of *yytext* rather than replacing it; the value in *yyleng* shall be adjusted accordingly.

int *yyless*(**int** *n*)
Retains *n* initial characters in *yytext*, NUL-terminated, and treats the remaining characters as if they had not been read; the value in *yyleng* shall be adjusted accordingly.

int *input*(**void**)
Returns the next character from the input, or zero on end-of-file. It shall obtain input from the stream pointer *yyin*, although possibly via an intermediate buffer. Thus, once scanning has begun, the effect of altering the value of *yyin* is undefined. The character read shall be removed from the input stream of the scanner without any processing by the scanner.

int *unput*(**int** *c*)
Returns the character `'c'` to the input; *yytext* and *yyleng* are undefined until the next expression is matched. The result of using *unput*( ) for more characters than have been input is unspecified.

The following functions shall appear only in the *lex* library accessible through the **–l l** operand; they can therefore be redefined by a conforming application:

int *yywrap*(**void**)
Called by *yylex*( ) at end-of-file; the default *yywrap*( ) shall always return 1. If the application requires *yylex*( ) to continue processing with another source of input, then the application

can include a function *yywrap*(), which associates another file with the external variable **FILE** * *yyin* and shall return a value of zero.

int *main*(**int** *argc*, **char** **argv*[ ])
Calls *yylex*() to perform lexical analysis, then exits. The user code can contain *main*() to perform application-specific operations, calling *yylex*() as applicable.

Except for *input*(), *unput*(), and *main*(), all external and static names generated by *lex* shall begin with the prefix **yy** or **YY**.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Conforming applications are warned that in the *Rules* section, an ERE without an action is not acceptable, but need not be detected as erroneous by *lex*. This may result in compilation or runtime errors.

The purpose of *input*() is to take characters off the input stream and discard them as far as the lexical analysis is concerned. A common use is to discard the body of a comment once the beginning of a comment is recognized.

The *lex* utility is not fully internationalized in its treatment of regular expressions in the *lex* source code or generated lexical analyzer. It would seem desirable to have the lexical analyzer interpret the regular expressions given in the *lex* source according to the environment specified when the lexical analyzer is executed, but this is not possible with the current *lex* technology. Furthermore, the very nature of the lexical analyzers produced by *lex* must be closely tied to the lexical requirements of the input language being described, which is frequently locale-specific anyway. (For example, writing an analyzer that is used for French text is not automatically useful for processing other languages.)

EXAMPLES

The following is an example of a *lex* program that implements a rudimentary scanner for a Pascal-like syntax:

```
%{
/* Need this for the call to atof() below. */
#include <math.h>
/* Need this for printf(), fopen(), and stdin below. */
#include <stdio.h>
%}

DIGIT    [0-9]
ID       [a-z][a-z0-9]*

%%

{DIGIT}+ {
    printf("An integer: %s (%d)\n", yytext,
        atoi(yytext));
    }
```



```
{DIGIT}+"."{DIGIT}*        {
    printf("A float: %s (%g)\n", yytext,
        atof(yytext));
    }
if|then|begin|end|procedure|function        {
    printf("A keyword: %s\n", yytext);
    }
{ID}    printf("An identifier: %s\n", yytext);
"+"|"-"|"*"|"/"        printf("An operator: %s\n", yytext);
"{"[^}\n]*"}"    /* Eat up one-line comments. */
[ \t\n]+        /* Eat up white space. */
.  printf("Unrecognized character: %s\n", yytext);
%%
int main(int argc, char *argv[])
{
    ++argv, --argc;  /* Skip over program name. */
    if (argc > 0)
        yyin = fopen(argv[0], "r");
    else
        yyin = stdin;
    yylex();
}
```

RATIONALE

Even though references to the C language are retained in this description, *lex* may be generalized to other languages, as was done at one time for EFL, the Extended FORTRAN Language. Since the *lex* input specification is essentially language-independent, versions of this utility could be written to produce Ada, Modula-2, or Pascal code, and there are known historical implementations that do so.

The current description of *lex* bypasses the issue of dealing with internationalized EREs in the *lex* source code or generated lexical analyzer. If it follows the model used by *awk* (the source code is assumed to be presented in the POSIX locale, but input and output are in the locale specified by the environment variables), then the tables in the lexical analyzer produced by *lex* would interpret EREs specified in the *lex* source in terms of the environment variables specified when *lex* was executed. The desired effect would be to have the lexical analyzer interpret the EREs given in the *lex* source according to the environment specified when the lexical analyzer is executed, but this is not possible with the current *lex* technology.

The description of octal and hexadecimal-digit escape sequences agrees with the ISO C standard use of escape sequences.

Earlier versions of this standard allowed for implementations with bytes other than eight bits, but this has been modified in this version.

There is no detailed output format specification. The observed behavior of *lex* under four different historical implementations was that none of these implementations consistently reported the line numbers for error and warning messages. Furthermore, there was a desire that *lex* be allowed to output additional diagnostic messages. Leaving message formats unspecified avoids these formatting questions and problems with internationalization.

Although the `%x` specifier for *exclusive* start conditions is not historical practice, it is believed to be a minor change to historical implementations and greatly enhances the usability of *lex* programs since it permits an application to obtain the expected functionality with fewer statements.

The **%array** and **%pointer** declarations were added as a compromise between historical systems. The System V-based *lex* copies the matched text to a *yytext* array. The *flex* program, supported in BSD and GNU systems, uses a pointer. In the latter case, significant performance improvements are available for some scanners. Most historical programs should require no change in porting from one system to another because the string being referenced is null-terminated in both cases. (The method used by *flex* in its case is to null-terminate the token in place by remembering the character that used to come right after the token and replacing it before continuing on to the next scan.) Multi-file programs with external references to *yytext* outside the scanner source file should continue to operate on their historical systems, but would require one of the new declarations to be considered strictly portable.

The description of EREs avoids unnecessary duplication of ERE details because their meanings within a *lex* ERE are the same as that for the ERE in this volume of POSIX.1-2024.

The reason for the undefined condition associated with text beginning with a <blank> or within `"%{"` and `"%}"` delimiter lines appearing in the *Rules* section is historical practice. Both the BSD and System V *lex* copy the indented (or enclosed) input in the *Rules* section (except at the beginning) to unreachable areas of the *yylex*() function (the code is written directly after a *break* statement). In some cases, the System V *lex* generates an error message or a syntax error, depending on the form of indented input.

The intention in breaking the list of functions into those that may appear in **lex.yy.c** *versus* those that only appear in **libl.a** is that only those functions in **libl.a** can be reliably redefined by a conforming application.

The descriptions of standard output and standard error are somewhat complicated because historical *lex* implementations chose to issue diagnostic messages to standard output (unless **−t** was given). POSIX.1-2024 allows this behavior, but leaves an opening for the more expected behavior of using standard error for diagnostics. Also, the System V behavior of writing the statistics when any table sizes are given is allowed, while BSD-derived systems can avoid it. The programmer can always precisely obtain the desired results by using either the **−t** or **−n** options.

The OPERANDS section does not mention the use of − as a synonym for standard input; not all historical implementations support such usage for any of the *file* operands.

A description of the *translation table* was deleted from early proposals because of its relatively low usage in historical applications.

The change to the definition of the *input*() function that allows buffering of input presents the opportunity for major performance gains in some applications.

The following examples clarify the differences between *lex* regular expressions and regular expressions appearing elsewhere in this volume of POSIX.1-2024. For regular expressions of the form `"r/x"`, the string matching *r* is always returned; confusion may arise when the beginning of *x* matches the trailing portion of *r*. For example, given the regular expression `"a*b/cc"` and the input `"aaabcc"`, *yytext* would contain the string `"aaab"` on this match. But given the regular expression `"x*/xy"` and the input `"xxxy"`, the token **xxx**, not **xx**, is returned by some implementations because **xxx** matches `"x*"`.

In the rule `"ab*/bc"`, the `"b*"` at the end of *r* extends *r*'s match into the beginning of the trailing context, so the result is unspecified. If this rule were `"ab/bc"`, however, the rule matches the text `"ab"` when it is followed by the text `"bc"`. In this latter case, the matching of *r*

cannot extend into the beginning of *x*, so the result is specified.

FUTURE DIRECTIONS

None.

SEE ALSO

c17, *ed*, *yacc*

XBD Chapter 5 (on page 113), Chapter 8 (on page 167), Chapter 9 (on page 179), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is marked as part of the C-Language Development Utilities option.

The obsolescent **−c** option is removed.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/14 is applied, removing text describing behavior on systems with bytes consisting of more than eight bits.

Issue 7

Austin Group Interpretation 1003.1-2001 #190 is applied, clarifying the requirements for generated code to conform to the ISO C standard.

Austin Group Interpretation 1003.1-2001 #191 is applied, clarifying the handling of C-language trigraphs and curly brace preprocessing tokens.

SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1453 is applied, changing the ENVIRONMENT VARIABLES section.

Austin Group Defect 1517 is applied, removing a reference to the **−c** option from the RATIONALE section.

NAME

link — call *link*( ) function

SYNOPSIS

XSI

```
link file1 file2
```

DESCRIPTION

The *link* utility shall perform the function call:

```
link(file1, file2);
```

A user may need appropriate privileges to invoke the *link* utility.

OPTIONS

None.

OPERANDS

The following operands shall be supported:

file1 The pathname of an existing file.

file2 The pathname of the new directory entry to be created.

STDIN

Not used.

INPUT FILES

Not used.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *link*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

None.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

ln, *unlink*

XBD Chapter 8 (on page 167)

XSH *link*( )

CHANGE HISTORY

First released in Issue 5.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

ln — link files

SYNOPSIS

```
ln [-fs] [-L|-P] source_file target_file
ln [-fs] [-L|-P] source_file... target_dir
```

DESCRIPTION

In the first synopsis form, the *ln* utility shall create a new directory entry at the destination path specified by the *target_file* operand. If the **−s** option is specified, a symbolic link shall be created with the contents specified by the *source_file* operand (which need not name an existing file); otherwise, a hard link shall be created to the file named by the *source_file* operand. This first synopsis form shall be assumed when the final operand does not name an existing directory; if more than two operands are specified and the final is not an existing directory, an error shall result.

In the second synopsis form, the *ln* utility shall create a new directory entry for each *source_file* operand, at a destination path in the existing directory named by *target_dir*. If the **−s** option is specified, a symbolic link shall be created with the contents specified by each *source_file* operand (which need not name an existing file); otherwise, a hard link shall be created to each file named by a *source_file* operand.

If the last operand specifies an existing file of a type not specified by the System Interfaces volume of POSIX.1-2024, the behavior is implementation-defined.

The corresponding destination path for each *source_file* shall be the concatenation of the target directory pathname, a <slash> character if the target directory pathname did not end in a <slash>, and the last pathname component of the *source_file*. The second synopsis form shall be assumed when the final operand names an existing directory.

For each *source_file*:

1. If the destination path exists and was created by a previous step, it is unspecified whether *ln* writes a diagnostic message to standard error, does nothing more with the current *source_file*, and goes on to any remaining *source_file*s; or continues processing the current *source_file*. If the destination path exists:
    a. If the **−f** option is not specified, *ln* shall write a diagnostic message to standard error, do nothing more with the current *source_file*, and go on to any remaining *source_file*s.
    b. If the destination path names the same directory entry as the current *source_file* *ln* shall write a diagnostic message to standard error, do nothing more with the current *source_file*, and go on to any remaining *source_file*s.
    c. Actions shall be performed equivalent to the *unlink*() function defined in the System Interfaces volume of POSIX.1-2024, called using the destination path as the *path* argument. If this fails for any reason, *ln* shall write a diagnostic message to standard error, do nothing more with the current *source_file*, and go on to any remaining *source_file*s.
2. If the **−s** option is specified, actions shall be performed equivalent to the *symlink*() function with *source_file* as the *path1* argument and the destination path as the *path2* argument. The *ln* utility shall do nothing more with *source_file* and shall go on to any remaining files.

3. If *source_file* is a symbolic link:

   a. If the **−P** option is in effect, actions shall be performed equivalent to the *linkat*() function with *source_file* as the *path1* argument, the destination path as the *path2* argument, AT_FDCWD as the *fd1* and *fd2* arguments, and zero as the *flag* argument.

   b. If the **−L** option is in effect, actions shall be performed equivalent to the *linkat*() function with *source_file* as the *path1* argument, the destination path as the *path2* argument, AT_FDCWD as the *fd1* and *fd2* arguments, and AT_SYMLINK_FOLLOW as the *flag* argument.

   The *ln* utility shall do nothing more with *source_file* and shall go on to any remaining files.

4. Actions shall be performed equivalent to the *link*() function defined in the System Interfaces volume of POSIX.1-2024 using *source_file* as the *path1* argument, and the destination path as the *path2* argument.

OPTIONS

The *ln* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−f Force existing destination pathnames to be removed to allow the link.

−L For each *source_file* operand that names a file of type symbolic link, create a hard link to the file referenced by the symbolic link.

−P For each *source_file* operand that names a file of type symbolic link, create a hard link to the symbolic link itself.

−s Create symbolic links instead of hard links. If the **−s** option is specified, the **−L** and **−P** options shall be silently ignored.

Specifying more than one of the mutually-exclusive options **−L** and **−P** shall not be considered an error. The last option specified shall determine the behavior of the utility (unless the **−s** option causes it to be ignored).

If the **−s** option is not specified and neither a **−L** nor a **−P** option is specified, it is implementation-defined which of the **−L** and **−P** options is used as the default.

OPERANDS

The following operands shall be supported:

source_file A pathname of a file to be linked. If the **−s** option is specified, no restrictions on the type of file or on its existence shall be made. If the **−s** option is not specified, whether a directory can be linked is implementation-defined.

target_file The pathname of the new directory entry to be created.

target_dir A pathname of an existing directory in which the new directory entries are created.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *ln*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

The CONSEQUENCES OF ERRORS section does not require *ln* **−f** *a b* to remove *b* if a subsequent link operation would fail.

Some historic versions of *ln* (including the one specified by the SVID) unlink the destination file, if it exists, by default. If the mode does not permit writing, these versions prompt for confirmation before attempting the unlink. In these versions the **−f** option causes *ln* not to attempt to prompt for confirmation.

This allows *ln* to succeed in creating links when the target file already exists, even if the file itself

is not writable (although the directory must be). Early proposals specified this functionality.

This volume of POSIX.1-2024 does not allow the *ln* utility to unlink existing destination paths by default for the following reasons:

- The *ln* utility has historically been used to provide locking for shell applications, a usage that is incompatible with *ln* unlinking the destination path by default. There was no corresponding technical advantage to adding this functionality.
- This functionality gave *ln* the ability to destroy the link structure of files, which changes the historical behavior of *ln*.
- This functionality is easily replicated with a combination of *rm* and *ln*.
- It is not historical practice in many systems; BSD and BSD-derived systems do not support this behavior. Unfortunately, whichever behavior is selected can cause scripts written expecting the other behavior to fail.
- It is preferable that *ln* perform in the same manner as the *link*( ) function, which does not permit the target to exist already.

This volume of POSIX.1-2024 retains the **−f** option to provide support for shell scripts depending on the SVID semantics. It seems likely that shell scripts would not be written to handle prompting by *ln* and would therefore have specified the **−f** option.

The **−f** option is an undocumented feature of many historical versions of the *ln* utility, allowing linking to directories. These versions require modification.

Early proposals of this volume of POSIX.1-2024 also required a **−i** option, which behaved like the **−i** options in *cp* and *mv*, prompting for confirmation before unlinking existing files. This was not historical practice for the *ln* utility and has been omitted.

The **−L** and **−P** options allow for implementing both common behaviors of the *ln* utility. Earlier versions of this standard did not specify these options and required the behavior now described for the **−L** option. Many systems by default or as an alternative provided a non-conforming *ln* utility with the behavior now described for the **−P** option. Since applications could not rely on *ln* following links in practice, the **−L** and **−P** options were added to specify the desired behavior for the application.

The **−L** and **−P** options are ignored when **−s** is specified in order to allow an alias to be created to alter the default behavior when creating hard links (for example, *alias ln='ln* **−L**'). They serve no purpose when **−s** is specified, since *source_file* is then just a string to be used as the contents of the created symbolic link and need not exist as a file.

The specification ensures that *ln* **a a** with or without the **−f** option will not unlink the file **a**. Earlier versions of this standard were unclear in this case.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

chmod, *find*, *pax*, *readlink*, *realpath*, *rm*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *link*( ), *unlink*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

The *ln* utility is updated to include symbolic link processing as defined in the IEEE P1003.2b draft standard.

Issue 7

Austin Group Interpretations 1003.1-2001 #164, #168, and #169 are applied.

SD5-XCU-ERN-27 is applied, adding a new paragraph to the RATIONALE.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The **–L** and **–P** options are added to make it implementation-defined whether the *ln* utility follows symbolic links.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0096 [136] is applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0113 [930] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1380 is applied, changing text using the term ``link'' in line with its updated definition.

Austin Group Defect 1457 is applied, adding *readlink* and *realpath* to the SEE ALSO section.

Austin Group Defect 1506 is applied, changing the EXIT STATUS section.

NAME

locale — get locale-specific information

SYNOPSIS

```
locale [-a|-m]
locale [-ck] name...
```

DESCRIPTION

The *locale* utility shall write information about the current locale environment, or all public locales, to the standard output. For the purposes of this section, a *public locale* is one provided by the implementation that is accessible to the application.

When *locale* is invoked without any arguments, it shall summarize the current locale environment for each locale category as determined by the settings of the environment variables defined in XBD Chapter 7 (on page 127).

When invoked with operands, it shall write values that have been assigned to the keywords in the locale categories, as follows:

- Specifying a keyword name shall select the named keyword and the category containing that keyword.
- Specifying a category name shall select the named category and all keywords in that category.

OPTIONS

The *locale* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−a Write information about all available public locales. The available locales shall include **POSIX**, representing the POSIX locale. The manner in which the implementation determines what other locales are available is implementation-defined.

−c Write the names of selected locale categories; see the STDOUT section. The **−c** option increases readability when more than one category is selected (for example, via more than one keyword name or via a category name). It is valid both with and without the **−k** option.

−k Write the names and values of selected keywords. The implementation may omit values for some keywords; see the OPERANDS section.

−m Write names of available charmaps; see XBD Section 6.1 (on page 117).

OPERANDS

The following operand shall be supported:

name The name of a locale category as defined in XBD Chapter 7 (on page 127), the name of a keyword in a locale category, or the reserved name **charmap**. The named category or keyword shall be selected for output. If a single *name* represents both a locale category name and a keyword name in the current locale, the results are unspecified. Otherwise, both category and keyword names can be specified as *name* operands, in any sequence. It is implementation-defined whether any keyword values are written for the categories *LC_CTYPE* and *LC_COLLATE*.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *locale*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

XSI The application shall ensure that the *LANG*, *LC_**, and *NLSPATH* environment variables specify the current locale environment to be written out; they shall be used if the **–a** option is not specified.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The *LANG* variable shall be written first using the format:

```
"LANG=%s\n", <value>
```

If *LANG* is not set or is an empty string, the value is the empty string.

If *locale* is invoked without any options or operands, the names and values of the *LC_** environment variables described in this volume of POSIX.1-2024 shall be written to the standard output, one variable per line, and each line using the following format. Only those variables set in the environment and not overridden by *LC_ALL* shall be written using this format:

```
"%s=%s\n", <variable_name>, <value>
```

The names of those *LC_** variables associated with locale categories defined in this volume of POSIX.1-2024 that are not set in the environment or are overridden by *LC_ALL* shall be written in the following format:

```
"%s=\"%s\"\n", <variable_name>, <implied value>
```

The **<***implied value***>** shall be the name of the locale that has been selected for that category by the implementation, based on the values in *LANG* and *LC_ALL*, as described in XBD Chapter 8 (on page 167).

The **<***value***>** and **<***implied value***>** shown above shall be properly quoted for possible later reentry to the shell. The **<***value***>** shall not be quoted using double-quotes (so that it can be distinguished by the user from the **<***implied value***>** case, which always requires double-quotes).

The *LC_ALL* variable shall be written last, using the first format shown above. If it is not set, it shall be written as:

```
"LC_ALL=\n"
```

If any arguments are specified:

1. If the **−a** option is specified, the names of all the public locales shall be written, each in the following format:

   ```
   "%s\n", <locale name>
   ```

2. If the **−c** option is specified, the names of all selected categories shall be written, each in the following format:

   ```
   "%s\n", <category name>
   ```

   If keywords are also selected for writing (see following items), the category name output shall precede the keyword output for that category.

   If the **−c** option is not specified, the names of the categories shall not be written; only the keywords, as selected by the **<***name***>** operand, shall be written.

3. If the **−k** option is specified, the names and values of selected keywords shall be written. If a value is non-numeric and is not a compound keyword value, it shall be written in the following format:

   ```
   "%s=\"%s\"\n", <keyword name>, <keyword value>
   ```

   If a value is a non-numeric compound keyword value, it shall either be written in the format:

   ```
   "%s=\"%s\"\n", <keyword name>, <keyword value>
   ```

   where the **<***keyword value***>** is a single string of values separated by **<**semicolon**>** characters, or it shall be written in the format:

   ```
   "%s=%s\n", <keyword name>, <keyword value>
   ```

   where the **<***keyword value***>** is encoded as a set of strings, each enclosed in double-quotation-marks, separated by **<**semicolon**>** characters.

   If the keyword was **charmap**, the name of the charmap (if any) that was specified via the *localedef* **−f** option when the locale was created shall be written, with the word **charmap** as **<***keyword name***>**.

   If a value is numeric, it shall be written in one of the following formats:

   ```
   "%s=%d\n", <keyword name>, <keyword value>
   ```

   ```
   "%s=%c%o\n", <keyword name>, <escape character>, <keyword value>
   ```

   ```
   "%s=%cx%x\n", <keyword name>, <escape character>, <keyword value>
   ```

   where the **<***escape character***>** is that identified by the **escape_char** keyword in the current locale; see XBD Section 7.3 (on page 128).

   Compound keyword values (list entries) shall be separated in the output by **<**semicolon**>** characters. When included in keyword values, the **<**semicolon**>**, **<**backslash**>**, double-quote, and any control character shall be preceded (escaped) with the escape character.

4. If the **–k** option is not specified, selected keyword values shall be written, each in the following format:

   `"%s\n", <keyword value>`

   If the keyword was **charmap**, the name of the charmap (if any) that was specified via the *localedef* **–f** option when the locale was created shall be written.

5. If the **–m** option is specified, then a list of all available charmaps shall be written, each in the format:

   `"%s\n", <charmap>`

   where **<***charmap***>** is in a format suitable for use as the option-argument to the *localedef* **–f** option.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All the requested information was found and output successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

If the *LANG* environment variable is not set or set to an empty value, or one of the *LC_** environment variables is set to an unrecognized value, the actual locales assumed (if any) are implementation-defined as described in XBD Chapter 8 (on page 167).

Implementations are not required to write out the actual values for keywords in the categories *LC_CTYPE* and *LC_COLLATE*; however, they must write out the categories (allowing an application to determine, for example, which character classes are available).

EXAMPLES

In the following examples, the assumption is that locale environment variables are set as follows:

```
LANG=locale_x
LC_COLLATE=locale_y
```

The command *locale* would result in the following output:

```
LANG=locale_x
LC_CTYPE="locale_x"
LC_COLLATE=locale_y
LC_TIME="locale_x"
LC_NUMERIC="locale_x"
LC_MONETARY="locale_x"
LC_MESSAGES="locale_x"
LC_ALL=
```


The order of presentation of the categories is not specified by this volume of POSIX.1-2024.

The command:

```
LC_ALL=POSIX locale -ck decimal_point
```

would produce:

```
LC_NUMERIC
decimal_point="."
```

The following command shows an application of *locale* to determine whether a user-supplied response is affirmative:

```
printf 'Prompt for response: '
read response
if printf "%s\n" "$response" | grep -Eq -- "$(locale yesexpr)"
then
    affirmative processing goes here
else
    non-affirmative processing goes here
fi
```

RATIONALE

The output for categories *LC_CTYPE* and *LC_COLLATE* has been made implementation-defined because there is a questionable value in having a shell script receive an entire array of characters. It is also difficult to return a logical collation description, short of returning a complete *localedef* source.

The **–m** option was included to allow applications to query for the existence of charmaps. The output is a list of the charmaps (implementation-supplied and user-supplied, if any) on the system.

The **–c** option was included for readability when more than one category is selected (for example, via more than one keyword name or via a category name). It is valid both with and without the **–k** option.

The **charmap** keyword, which returns the name of the charmap (if any) that was used when the current locale was created, was included to allow applications needing the information to retrieve it.

According to XBD Section 6.1 (on page 117), the standard requires that all supported locales must have the same encoding for <period> and <slash>, because these two characters are used within the locale-independent pathname resolution sequence. Therefore, it would be an error if *locale* **–a** listed both ASCII and EBCDIC-based locales, since those two encodings do not share the same representation for either <period> or <slash>. Any system that supports both environments would be expected to provide two POSIX locales, one in either codeset, where only the locales appropriate to the current environment can be visible at a time. In an XSI-compliant implementation, the *dd* utility is the only portable means for performing conversions between the two character sets.

FUTURE DIRECTIONS

None.

SEE ALSO

localedef

XBD Section 6.1 (on page 117), Chapter 7 (on page 127), Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

The normative text is reworded to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/30 is applied, correcting an editorial error in the STDOUT section.

Issue 7

Austin Group Interpretations 1003.1-2001 #017, #021, and #088 are applied, clarifying the standard output for the **−k** option when *LANG* is not set or is an empty string.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0097 [291] is applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0114 [941] is applied.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1262 is applied, changing the EXAMPLES section.

NAME

localedef — define locale environment

SYNOPSIS

```
localedef [-c] [-f charmap] [-i sourcefile] [-u code_set_name] name
```

DESCRIPTION

The *localedef* utility shall convert source definitions for locale categories into a format usable by the functions and utilities whose operational behavior is determined by the setting of the locale environment variables defined in XBD Chapter 7 (on page 127). It is implementation-defined whether users have the capability to create new locales, in addition to those supplied by the implementation. If the symbolic constant POSIX2_LOCALEDEF is defined, the system supports

XSI the creation of new locales. On XSI-conformant systems, the symbolic constant POSIX2_LOCALEDEF shall be defined.

The utility shall read source definitions for one or more locale categories belonging to the same locale from the file named in the **–i** option (if specified) or from standard input.

The *name* operand identifies the target locale. The utility shall support the creation of *public*, or generally accessible locales, as well as *private*, or restricted-access locales. Implementations may restrict the capability to create or modify public locales to users with appropriate privileges.

Each category source definition shall be identified by the corresponding environment variable name and terminated by an **END** *category-name* statement. The following categories shall be supported. In addition, the input may contain source for implementation-defined categories.

LC_CTYPE Defines character classification and case conversion.

LC_COLLATE
Defines collation rules.

LC_MONETARY
Defines the format and symbols used in formatting of monetary information.

LC_NUMERIC
Defines the decimal delimiter, grouping, and grouping symbol for non-monetary numeric editing.

LC_TIME Defines the format and content of date and time information.

LC_MESSAGES
Defines the format and values of affirmative and negative responses.

If the *LC_COLLATE* category defines a collation sequence that does not have a total ordering of all characters, *localedef* shall write a warning message to standard error and, if the exit status would otherwise have been zero, shall exit with status 1.

OPTIONS

The *localedef* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–c Create permanent output even if warning messages have been issued.

–f *charmap* Specify the pathname of a file containing a mapping of character symbols and collating element symbols to actual character encodings. The format of the *charmap* is described in XBD Section 6.4 (on page 121). The application shall ensure that this option is specified if symbolic names (other than collating symbols defined in a **collating-symbol** keyword) are used. If the **–f** option is not present, an implementation-defined character mapping shall be used.

–i *inputfile* The pathname of a file containing the source definitions. If this option is not present, source definitions shall be read from standard input. The format of the *inputfile* is described in XBD Section 7.3 (on page 128).

–u *code_set_name*
Specify the name of a codeset used as the target mapping of character symbols and collating element symbols whose encoding values are defined in terms of the ISO/IEC 10646: 2020 standard position constant values.

OPERANDS

The following operand shall be supported:

name Identifies the locale; see XBD Chapter 7 (on page 127) for a description of the use of this name. If the name contains one or more <slash> characters, *name* shall be interpreted as a pathname where the created locale definitions shall be stored. If *name* does not contain any <slash> characters, the interpretation of the name is implementation-defined and the locale shall be public. The ability to create public locales in this way may be restricted to users with appropriate privileges. (As a consequence of specifying one *name*, although several categories can be processed in one execution, only categories belonging to the same locale can be processed.)

STDIN

Unless the **–i** option is specified, the standard input shall be a text file containing one or more locale category source definitions, as described in XBD Section 7.3 (on page 128). When lines are continued using the escape character mechanism, there is no limit to the length of the accumulated continued line.

INPUT FILES

The character set mapping file specified as the *charmap* option-argument is described in XBD Section 6.4 (on page 121). If a locale category source definition contains a **copy** statement, as defined in XBD Chapter 7 (on page 127), and the **copy** statement names a valid, existing locale, then *localedef* shall behave as if the source definition had contained a valid category source definition for the named locale.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *localedef*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
(This variable has no affect on *localedef*; the POSIX locale is used for this category.)

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files). This variable has no affect on the processing of *localedef* input data; the POSIX locale is used for this purpose, regardless of the value of this variable.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

102762 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

102763 **ASYNCHRONOUS EVENTS**

102764 Default.

102765 **STDOUT**

102766 The utility shall report all categories successfully processed, in an unspecified format.

102767 **STDERR**

102768 The standard error shall be used only for diagnostic messages.

102769 **OUTPUT FILES**

102770 The format of the created output is unspecified. If the *name* operand does not contain a <slash>,
102771 the existence of an output file for the locale is unspecified.

102772 **EXTENDED DESCRIPTION**

102773 When the **−u** option is used, the *code_set_name* option-argument shall be interpreted as an
102774 implementation-defined name of a codeset to which the ISO/IEC 10646:2020 standard position
102775 constant values shall be converted via an implementation-defined method. Both the
102776 ISO/IEC 10646:2020 standard position constant values and other formats (decimal,
102777 hexadecimal, or octal) shall be valid as encoding values within the *charmap* file. The codeset
102778 represented by the implementation-defined name can be any codeset that is supported by the
102779 implementation.

102780 When conflicts occur between the *charmap* specification of <*code_set_name*>, <*mb_cur_max*>, or
102781 <*mb_cur_min*> and the implementation-defined interpretation of these respective items for the
102782 codeset represented by the **−u** option-argument *code_set_name*, the result is unspecified.

102783 When conflicts (including omissions) occur between the *charmap* encoding values specified for
102784 symbolic names of characters of the portable character set and the implementation-defined
102785 assignment of character encoding values, the result is unspecified. If the result is that *localedef*
102786 creates the specified locale, any attempted use of that locale by an application or utility results in
102787 undefined behavior.

102788 If a non-printable character in the *charmap* has a width specified that is not **−1**, the result is
102789 undefined.

102790 **EXIT STATUS**

102791 The following exit values shall be returned:

102792 0 No errors occurred and the locales were successfully created.

102793 1 Warnings occurred and the locales were successfully created.

102794 2 The locale specification exceeded implementation limits or the coded character set or sets
102795 used were not supported by the implementation, and no locale was created.

102796 3 The capability to create new locales is not supported by the implementation.

102797 >3 Warnings or errors occurred and no output was created.

102798 **CONSEQUENCES OF ERRORS**

102799 If an error is detected, no permanent output shall be created.

102800 If warnings occur, permanent output shall be created if the **−c** option was specified. The
102801 following conditions shall cause warning messages to be issued:

102802 • If a symbolic name not found in the *charmap* file is used for the descriptions of the
102803 *LC_CTYPE* or *LC_COLLATE* categories (for other categories, this shall be an error
102804 condition).

- If the number of operands to the **order** keyword exceeds the {COLL_WEIGHTS_MAX} limit.
- If optional keywords not supported by the implementation are present in the source.

Other implementation-defined conditions may also cause warnings.

APPLICATION USAGE

The *charmap* definition is optional, and is contained outside the locale definition. This allows both completely self-defined source files, and generic sources (applicable to more than one codeset). To aid portability, all *charmap* definitions must use the same symbolic names for the portable character set. As explained in XBD Section 6.4 (on page 121), it is implementation-defined whether or not users or applications can provide additional character set description files. Therefore, the **–f** option might be operable only when an implementation-defined *charmap* is named.

EXAMPLES

None.

RATIONALE

The output produced by the *localedef* utility is implementation-defined. The *name* operand is used to identify the specific locale. (As a consequence, although several categories can be processed in one execution, only categories belonging to the same locale can be processed.)

When conflicts (including omissions) occur between the *charmap* encoding values specified for symbolic names of characters of the portable character set and the implementation-defined assignment of character encoding values, it is recommended that *localedef* treats this as an error in order to prevent the undefined behavior that results if *localedef* creates the specified locale and an application or utility attempts to use it.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

locale

XBD Section 6.4 (on page 121), Chapter 7 (on page 127), Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 6

The **–u** option is added, as specified in the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/15 is applied, rewording text in the OPERANDS section describing the ability to create public locales.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/16 is applied, making the text consistent with the descriptions of **WIDTH** and **WIDTH_DEFAULT** in the Base Definitions volume of POSIX.1-2024.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1070 is applied, requiring that *localedef* issues a warning if the *LC_COLLATE* category defines a collation sequence that does not have a total ordering of all characters.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1609 is applied, clarifying the behavior when conflicts (including omissions) occur between the *charmap* encoding values specified for symbolic names of characters of the portable character set and the implementation-defined assignment of character encoding values.

NAME

logger — log messages

SYNOPSIS

```
logger [-i] [-f file] [-p priority] [-t tag] [string...]
```

DESCRIPTION

The *logger* utility shall send messages to an implementation-defined logging facility, which may log them in an implementation-defined system log, write them to the system console, forward them to a list of users, or forward them to the logging facility on another host over the network. Each logged message shall include a message header and a message body. The message header shall contain at least a timestamp and a tag string.

If one or more *string* operands are specified, they shall be logged; otherwise, the message bodies to be logged shall be read from standard input if no **−f** option is specified, or from the specified file if the **−f** option is present.

It is implementation-defined whether messages written in locales other than the POSIX locale are effective.

OPTIONS

The *logger* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−f *file* Read the log message bodies from *file* instead of standard input.

−i Log the process ID of the *logger* process with each message.

−p *priority* Log the message with priority set to *priority*. The priority is specified as a *facility*.*level* pair. The following values for *facility* shall be supported:

user	Messages generated by arbitrary processes.
local0	Reserved for local use.
local1	Reserved for local use.
local2	Reserved for local use.
local3	Reserved for local use.
local4	Reserved for local use.
local5	Reserved for local use.
local6	Reserved for local use.
local7	Reserved for local use.

The following values for *level* shall be supported:

emerg	A panic condition.
alert	A condition that should be corrected immediately, such as a corrupted system database.
crit	Critical conditions, such as hard device errors.
err	Errors.
warning	Warning messages.

102897 notice Conditions that are not error conditions, but that may require special
102898 handling.

102899 info Informational messages.

102900 debug Messages that contain information normally of use only when
102901 debugging a program.

102902 If the **–p** option is not specified, the priority shall be `user.notice`.

102903 **–t** *tag* Use the string *tag* as the tag string in the message header. The default tag is
102904 unspecified.

102905 **OPERANDS**

102906 The following operand shall be supported:

102907 *string* One of the string arguments whose contents are concatenated together, in the order
102908 specified, separated by single <space> characters.

102909 **STDIN**

102910 The standard input shall be used if no *string* operands are specified and either the **–f** option is
102911 not specified or the **–f** option is specified with a *file* option-argument of `'-'` and the
102912 implementation treats the `'-'` as meaning standard input. Otherwise, the standard input shall
102913 not be used. See the INPUT FILES section.

102914 Each non-empty line shall be logged as a separate message. It is unspecified whether an empty
102915 line is also logged as a separate message.

102916 **INPUT FILES**

102917 The input files shall be text files.

102918 **ENVIRONMENT VARIABLES**

102919 The following environment variables shall affect the execution of *logger*:

102920 *LANG* Provide a default value for the internationalization variables that are unset or null.
102921 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
102922 variables used to determine the values of locale categories.)

102923 *LC_ALL* If set to a non-empty string value, override the values of all the other
102924 internationalization variables.

102925 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
102926 characters (for example, single-byte as opposed to multi-byte characters in
102927 arguments).

102928 *LC_MESSAGES*
102929 Determine the locale that should be used to affect the format and contents of
102930 diagnostic messages written to standard error. (This means diagnostics from *logger*
102931 to the user or application, not diagnostic messages that the user is sending to the
102932 system administrator.)

102933 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

102934 **ASYNCHRONOUS EVENTS**

102935 Default.

102936 **STDOUT**

102937 Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

Unspecified.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

This utility allows logging of information for later use by a system administrator or programmer in determining why non-interactive utilities have failed. The locations of the saved messages, their format, and retention period are all unspecified. There is no method for a conforming application to read messages, once written.

EXAMPLES

A batch application, running non-interactively, tries to read a configuration file and fails; it may attempt to notify the system administrator with:

```
logger myname: unable to read file foo. [timestamp]
```

RATIONALE

The standard developers believed strongly that some method of alerting administrators to errors was necessary. The obvious example is a batch utility, running non-interactively, that is unable to read its configuration files or that is unable to create or write its results file. However, the standard developers did not wish to define the format or delivery mechanisms as they have historically been (and will probably continue to be) very system-specific, as well as involving functionality clearly outside the scope of this volume of POSIX.1-2024.

The text with *LC_MESSAGES* about diagnostic messages means diagnostics from *logger* to the user or application, not diagnostic messages that the user is sending to the system administrator.

Multiple *string* arguments are allowed, similar to *echo*, for ease-of-use.

Like the utilities *mailx* and *lp*, *logger* is admittedly difficult to test. This was not deemed sufficient justification to exclude these utilities from this volume of POSIX.1-2024. It is also arguable that they are, in fact, testable, but that the tests themselves are not portable.

FUTURE DIRECTIONS

None.

SEE ALSO

lp, *mailx*, *write*

XBD Chapter 8 (on page 167)

CHANGE HISTORY

First released in Issue 4.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 917 is applied, adding the **−f**, **−i**, **−p**, and **−t** options, and specifying the behavior when *logger* is executed with no operands.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

logname — return the user's login name

SYNOPSIS

```
logname
```

DESCRIPTION

The *logname* utility shall write the user's login name to standard output. The login name shall be the string that would be returned by the *getlogin*() function defined in the System Interfaces volume of POSIX.1-2024. Under the conditions where the *getlogin*() function would fail, the *logname* utility shall write a diagnostic message to standard error and exit with a non-zero exit status.

OPTIONS

None.

OPERANDS

None.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *logname*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The *logname* utility output shall be a single line consisting of the user's login name:

```
"%s\n", <login name>
```

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *logname* utility explicitly ignores the *LOGNAME* environment variable because environment changes could produce erroneous results.

EXAMPLES

None.

RATIONALE

The **passwd** file is not listed as required because the implementation may have other means of mapping login names.

FUTURE DIRECTIONS

None.

SEE ALSO

id, *who*

XBD Chapter 8 (on page 167)

XSH *getlogin*( )

CHANGE HISTORY

First released in Issue 2.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

lp — send files to a printer

SYNOPSIS

```
lp [-c] [-d dest] [-n copies] [-msw] [-o option]... [-t title] [file...]
```

DESCRIPTION

The *lp* utility shall copy the input files to an output destination in an unspecified manner. The default output destination should be to a hardcopy device, such as a printer or microfilm recorder, that produces non-volatile, human-readable documents. If such a device is not available to the application, or if the system provides no such device, the *lp* utility shall exit with a non-zero exit status.

The actual writing to the output device may occur some time after the *lp* utility successfully exits. During the portion of the writing that corresponds to each input file, the implementation shall guarantee exclusive access to the device.

The *lp* utility shall associate a unique *request ID* with each request.

Normally, a banner page is produced to separate and identify each print job. This page may be suppressed by implementation-defined conditions, such as an operator command or one of the **−o** *option* values.

OPTIONS

The *lp* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−c Exit only after further access to any of the input files is no longer required. The application can then safely delete or modify the files without affecting the output operation. Normally, files are not copied, but are linked whenever possible. If the **−c** option is not given, then the user should be careful not to remove any of the files before the request has been printed in its entirety. It should also be noted that in the absence of the **−c** option, any changes made to the named files after the request is made but before it is printed may be reflected in the printed output. On some implementations, **−c** may be on by default.

−d *dest* Specify a string that names the destination (*dest*). If *dest* is a printer, the request shall be printed only on that specific printer. If *dest* is a class of printers, the request shall be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, and so on), requests for specific destinations need not be accepted. Destination names vary between systems.

If **−d** is not specified, and neither the *LPDEST* nor *PRINTER* environment variable is set, an unspecified destination is used. The **−d** *dest* option shall take precedence over *LPDEST*, which in turn shall take precedence over *PRINTER*. Results are undefined when *dest* contains a value that is not a valid destination name.

−m Send mail (see *mailx*) after the files have been printed. By default, no mail is sent upon normal completion of the print request.

−n *copies* Write *copies* number of copies of the files, where *copies* is a positive decimal integer. The methods for producing multiple copies and for arranging the multiple copies when multiple *file* operands are used are unspecified, except that each file shall be output as an integral whole, not interleaved with portions of other files.

103096 **−o** *option* Specify printer-dependent or class-dependent *option*s. Several such *option*s may be
103097 collected by specifying the **−o** option more than once.

103098 **−s** Suppress messages from *lp*.

103099 **−t** *title* Write *title* on the banner page of the output.

103100 **−w** Write a message on the user's terminal after the files have been printed. If the user
103101 is not logged in, then mail shall be sent instead.

103102 **OPERANDS**

103103 The following operand shall be supported:

103104 *file* A pathname of a file to be output. If no *file* operands are specified, or if a *file*
103105 operand is `'−'`, the standard input shall be used. If a *file* operand is used, but the
103106 **−c** option is not specified, the process performing the writing to the output device
103107 may have user and group permissions that differ from that of the process invoking
103108 *lp*.

103109 **STDIN**

103110 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is `'−'`.
103111 See the INPUT FILES section.

103112 **INPUT FILES**

103113 The input files shall be text files.

103114 **ENVIRONMENT VARIABLES**

103115 The following environment variables shall affect the execution of *lp*:

103116 *LANG* Provide a default value for the internationalization variables that are unset or null.
103117 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
103118 variables used to determine the values of locale categories.)

103119 *LC_ALL* If set to a non-empty string value, override the values of all the other
103120 internationalization variables.

103121 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
103122 characters (for example, single-byte as opposed to multi-byte characters in
103123 arguments and input files).

103124 *LC_MESSAGES*
103125 Determine the locale that should be used to affect the format and contents of
103126 diagnostic messages written to standard error and informative messages written to
103127 standard output.

103128 *LC_TIME* Determine the format and contents of date and time strings displayed in the *lp*
103129 banner page, if any.

103130 *LPDEST* Determine the destination. If the *LPDEST* environment variable is not set, the
103131 *PRINTER* environment variable shall be used. The **−d** *dest* option takes precedence
103132 over *LPDEST*. Results are undefined when **−d** is not specified and *LPDEST*
103133 contains a value that is not a valid destination name.

103134 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

103135 *PRINTER* Determine the output device or destination. If the *LPDEST* and *PRINTER*
103136 environment variables are not set, an unspecified output device is used. The **−d**
103137 *dest* option and the *LPDEST* environment variable shall take precedence over
103138 *PRINTER*. Results are undefined when **−d** is not specified, *LPDEST* is unset, and
103139 *PRINTER* contains a value that is not a valid device or destination name.

TZ Determine the timezone used to calculate date and time strings displayed in the *lp* banner page, if any. If *TZ* is unset or null, an unspecified default timezone shall be used.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The *lp* utility shall write a *request ID* to the standard output, unless **–s** is specified. The format of the message is unspecified. The request ID can be used on systems supporting the historical *cancel* and *lpstat* utilities.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All input files were processed successfully.

>0 No output device was available, or an error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *pr* and *fold* utilities can be used to achieve reasonable formatting for the implementation's default page size.

A conforming application can use one of the *file* operands only with the **–c** option or if the file is publicly readable and guaranteed to be available at the time of printing. This is because POSIX.1-2024 gives the implementation the freedom to queue up the request for printing at some later time by a different process that might not be able to access the file.

EXAMPLES

1. To print file *file*:

   ```
   lp –c file
   ```

2. To print multiple files with headers:

   ```
   pr file1 file2 | lp
   ```

RATIONALE

The *lp* utility was designed to be a basic version of a utility that is already available in many historical implementations. The standard developers considered that it should be implementable simply as:

```
cat "$@" > /dev/lp
```

after appropriate processing of options, if that is how the implementation chose to do it and if exclusive access could be granted (so that two users did not write to the device simultaneously). Although in the future the standard developers may add other options to this utility, it should always be able to execute with no options or operands and send the standard input to an

unspecified output device.

This volume of POSIX.1-2024 makes no representations concerning the format of the printed output, except that it must be ``human-readable'' and ``non-volatile''. Thus, writing by default to a disk or tape drive or a display terminal would not qualify. (Such destinations are not prohibited when **−d** *dest*, *LPDEST*, or *PRINTER* are used, however.)

This volume of POSIX.1-2024 is worded such that a ``print job'' consisting of multiple input files, possibly in multiple copies, is guaranteed to print so that any one file is not intermixed with another, but there is no statement that all the files or copies have to print out together.

The **−c** option may imply a spooling operation, but this is not required. The utility can be implemented to wait until the printer is ready and then wait until it is finished. Because of that, there is no attempt to define a queuing mechanism (priorities, classes of output, and so on).

On some historical systems, the request ID reported on the STDOUT can be used to later cancel or find the status of a request using utilities not defined in this volume of POSIX.1-2024.

Although the historical System V *lp* and BSD *lpr* utilities have provided similar functionality, they used different names for the environment variable specifying the destination printer. Since the name of the utility here is *lp*, *LPDEST* (used by the System V *lp* utility) was given precedence over *PRINTER* (used by the BSD *lpr* utility). Since environments of users frequently contain one or the other environment variable, the *lp* utility is required to recognize both. If this was not done, many applications would send output to unexpected output devices when users moved from system to system.

Some have commented that *lp* has far too little functionality to make it worthwhile. Requests have proposed additional options or operands or both that added functionality. The requests included:

- Wording *requiring* the output to be ``hardcopy''
- A requirement for multiple printers
- Options for supporting various page-description languages

Given that a compliant system is not required to even have a printer, placing further restrictions upon the behavior of the printer is not useful. Since hardcopy format is so application-dependent, it is difficult, if not impossible, to select a reasonable subset of functionality that should be required on all compliant systems.

The term *unspecified* is used in this section in lieu of *implementation-defined* as most known implementations would not be able to make definitive statements in their conformance documents; the existence and usage of printers is very dependent on how the system administrator configures each individual system.

Since the default destination, device type, queuing mechanisms, and acceptable forms of input are all unspecified, usage guidelines for what a conforming application can do are as follows:

- Use the command in a pipeline, or with **−c**, so that there are no permission problems and the files can be safely deleted or modified.
- Limit output to text files of reasonable line lengths and printable characters and include no device-specific formatting information, such as a page description language. The meaning of ``reasonable'' in this context can only be answered as a quality-of-implementation issue, but it should be apparent from historical usage patterns in the industry and the locale. The *pr* and *fold* utilities can be used to achieve reasonable formatting for the default page size of the implementation.

Alternatively, the application can arrange its installation in such a way that it requires the system administrator or operator to provide the appropriate information on *lp* options and environment variable values.

At a minimum, having this utility in this volume of POSIX.1-2024 tells the industry that conforming applications require a means to print output and provides at least a command name and *LPDEST* routing mechanism that can be used for discussions between vendors, application developers, and users. The use of ``should'' in the DESCRIPTION of *lp* clearly shows the intent of the standard developers, even if they cannot mandate that all systems (such as laptops) have printers.

This volume of POSIX.1-2024 does not specify what the ownership of the process performing the writing to the output device may be. If **−c** is not used, it is unspecified whether the process performing the writing to the output device has permission to read *file* if there are any restrictions in place on who may read *file* until after it is printed. Also, if **−c** is not used, the results of deleting *file* before it is printed are unspecified.

FUTURE DIRECTIONS

None.

SEE ALSO

mailx

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the requirement to associate a unique request ID, and the normal generation of a banner page is added.
- In the OPTIONS section:
  - The **−d** *dest* description is expanded, but references to *lpstat* are removed.
  - The **−m**, **−o**, **−s**, **−t**, and **−w** options are added.
- In the ENVIRONMENT VARIABLES section, *LC_TIME* may now affect the execution.
- The STDOUT section is added.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

ls — list directory contents

SYNOPSIS

XSI
```
ls [-ikqrs] [-glno] [-A|-a] [-C|-m|-x|-1] \
    [-F|-p] [-H|-L] [-R|-d] [-S|-f|-t] [-c|-u] [file...]
```

DESCRIPTION

For each operand that names a file of a type other than directory or symbolic link to a directory, *ls* shall write the name of the file as well as any requested, associated information. For each operand that names a file of type directory, *ls* shall write the names of files contained within the directory as well as any requested, associated information. Filenames beginning with a <period> (`'.'`) and any associated information shall not be written out unless explicitly referenced, the **–A** or **–a** option is supplied, or an implementation-defined condition causes them to be written. If one or more of the **–d**, **–F**, or **–l** options are specified, and neither the **–H** nor the **–L** option is specified, for each operand that names a file of type symbolic link to a directory, *ls* shall write the name of the file as well as any requested, associated information. If none of the **–d**, **–F**, or **–l** options are specified, or the **–H** or **–L** options are specified, for each operand that names a file of type symbolic link to a directory, *ls* shall write the names of files contained within the directory as well as any requested, associated information. In each case where the names of files contained within a directory are written, if the directory contains any symbolic links then *ls* shall evaluate the file information and file type to be those of the symbolic link itself, unless the **–L** option is specified.

If no operands are specified, *ls* shall behave as if a single operand of dot (`'.'`) had been specified. If more than one operand is specified, *ls* shall write non-directory operands first; it shall sort directory and non-directory operands separately according to the collating sequence in the current locale.

Whenever *ls* sorts filenames or pathnames according to the collating sequence in the current locale, if this collating sequence does not have a total ordering of all characters (see XBD Section 7.3.2, on page 139), then any filenames or pathnames that collate equally shall be further compared byte-by-byte using the collating sequence for the POSIX locale.

The *ls* utility shall detect infinite loops; that is, entering a previously visited directory that is an ancestor of the last file encountered. When it detects an infinite loop, *ls* shall write a diagnostic message to standard error and shall either recover its position in the hierarchy or terminate.

OPTIONS

The *ls* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–A Write out all directory entries, including those whose names begin with a <period> (`'.'`) but excluding the entries dot and dot-dot (if they exist).

–C Write multi-text-column output with entries sorted down the columns, according to the collating sequence. The number of text columns and the column separator characters are unspecified, but should be adapted to the nature of the output device. This option disables long format output.

Note: Since the output from this option may use separator characters that include characters that might appear in filenames (in addition to the problems related to <newline>s in filenames), **–C** should not be used when filenames might be extracted from the output by a script.

−F Do not follow symbolic links named as operands unless the **−H** or **−L** options are specified. Write a <slash> (`'/'`) immediately after each pathname that is a directory, an <asterisk> (`'*'`) after each that is executable, a <vertical-line> (`'|'`) after each that is a FIFO, an <equals-sign> (`'='`) after each that is a socket, and a <commercial-at> (`'@'`) after each that is a symbolic link. For other file types, other symbols may be written.

−H Evaluate the file information and file type for symbolic links specified on the command line to be those of the file referenced by the link, and not the link itself; however, *ls* shall write the name of the link itself and not the file referenced by the link.

−L Evaluate the file information and file type for all symbolic links (whether named on the command line or encountered in a file hierarchy) to be those of the file referenced by the link, and not the link itself; however, *ls* shall write the name of the link itself and not the file referenced by the link. When **−L** is used with **−l**, write the contents of symbolic links in the long format (see the STDOUT section).

−R Recursively list subdirectories encountered. Subdirectories with filenames beginning with a <period> (`'.'`) shall be recursively listed if and only if the subdirectory name was included in the filenames listed for the containing directory. When a symbolic link to a directory is encountered, the directory shall not be recursively listed unless the **−L** option is specified. The use of **−R** with **−d** or **−f** produces unspecified results.

−S Sort with the primary key being file size (in decreasing order) and the secondary key being filename in the collating sequence (in increasing order). For a symbolic link, the size used as the sort key is that of the symbolic link itself, unless *ls* is evaluating its file information to be that of the file referenced by the link (see the **−H** and **−L** options).

−a Write out all directory entries, including those whose names begin with a <period> (`'.'`).

−c Use time of last modification of the file status information (see XBD **<sys/stat.h>**) instead of last modification of the file itself for sorting (**−t**) or writing (**−l**).

−d Do not follow symbolic links named as operands unless the **−H** or **−L** options are specified. Do not treat directories differently than other types of files. The use of **−d** with **−R** or **−f** produces unspecified results.

−f List the entries in directory operands in the order they appear in the directory. The behavior for non-directory operands is unspecified. This option shall turn on **−a**. When **−f** is specified, any occurrences of the **−r**, **−S**, and **−t** options shall be ignored and any occurrences of the **−A**, **−g**, **−l**, **−n**, **−o**, and **−s** options may be ignored. The use of **−f** with **−R** or **−d** produces unspecified results. XSI

−g Turn on the **−l** (ell) option, but disable writing the file's owner name or number. Disable the **−C**, **−m**, and **−x** options. XSI

−i For each file, write the file's file serial number (see *stat*( ) in the System Interfaces volume of POSIX.1-2024).

−k Set the block size for the **−s** option and the per-directory block count written for the **−l**, **−n**, **−s**, **−g**, and **−o** options (see the STDOUT section) to 1 024 bytes. XSI

–l (The letter ell.) Do not follow symbolic links named as operands unless the **–H** or **–L** options are specified. Write out in long format (see the STDOUT section). Disable the **–C**, **–m**, and **–x** options.

–m Stream output format; list pathnames across the page, separated by a <comma> character followed by a <space> character. Use a <newline> character as the list terminator and after the separator sequence when there is not room on a line for the next list entry. This option disables long format output.

–n Turn on the **–l** (ell) option, but when writing the file's owner or group, write the file's numeric UID or GID rather than the user or group name, respectively. Disable the **–C**, **–m**, and **–x** options.

XSI **–o** Turn on the **–l** (ell) option, but disable writing the file's group name or number. Disable the **–C**, **–m**, and **–x** options.

–p Write a <slash> (`'/'`) after each pathname if that file is a directory.

–q Force each instance of non-printable filename characters (including <newline>, <tab>, and other control characters) to be written as the <question-mark> (`'?'`) character. Implementations may provide this option by default if the output is to a terminal device.

–r Reverse the order of the sort to get reverse collating sequence, oldest first, or smallest file size first depending on the other options given.

–s Indicate the total number of file system blocks consumed by each file displayed. If the **–k** option is also specified, the block size shall be 1 024 bytes; otherwise, the block size is implementation-defined.

–t Sort with the primary key being time modified (most recently modified first) and the secondary key being filename in the collating sequence. For a symbolic link, the time used as the sort key is that of the symbolic link itself, unless *ls* is evaluating its file information to be that of the file referenced by the link (see the **–H** and **–L** options).

–u Use time of last access (see XBD **<sys/stat.h>**) instead of last modification of the file for sorting (**–t**) or writing (**–l**).

–x The same as **–C**, except that the multi-text-column output is produced with entries sorted across, rather than down, the columns. This option disables long format output.

–1 (The numeric digit one.) Force output to be one entry per line. This option does
XSI not disable long format output. (Long format output is enabled by **–g**, **–l** (ell), **–n**,
XSI and **–o**; and disabled by **–C**, **–m**, and **–x**.)

XSI If an option that enables long format output (**–g**, **–l** (ell), **–n**, and **–o**) is given with an option that disables long format output (**–C**, **–m**, and **–x**), this shall not be considered an error. The last of these options specified shall determine whether long format output is written.

If **–R**, **–d**, or **–f** are specified, the results of specifying these mutually-exclusive options are specified by the descriptions of these options above. If more than one of any of the other options shown in the SYNOPSIS section in mutually-exclusive sets are given, this shall not be considered an error; the last option specified in each set shall determine the output.

Note that if **–t** is specified, **–c** and **–u** are not only mutually-exclusive with each other, they are also mutually-exclusive with **–S** when determining sort order. But even if **–S** is specified after all occurrences of **–c**, **–t**, and **–u**, the last use of **–c** or **–u** determines the timestamp printed when

103397 producing long format output.

103398 **OPERANDS**

103399 The following operand shall be supported:

103400 *file* A pathname of a file to be written. If the file specified is not found, a diagnostic
103401 message shall be output on standard error.

103402 **STDIN**

103403 Not used.

103404 **INPUT FILES**

103405 None.

103406 **ENVIRONMENT VARIABLES**

103407 The following environment variables shall affect the execution of *ls*:

103408 *COLUMNS* Override the system-selected horizontal display line size, used to determine the
103409 column position width for writing multiple text-column output. See XBD Chapter
103410 8 (on page 167) for valid values and results when it is unset or null. The *ls* utility
103411 shall use this value to calculate how many pathname text columns to write (see
103412 **–C**). The column width chosen to write the names of files in any given directory
103413 shall be constant. Filenames shall not be truncated to fit into the multiple text-
103414 column output.

103415 *LANG* Provide a default value for the internationalization variables that are unset or null.
103416 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
103417 variables used to determine the values of locale categories.)

103418 *LC_ALL* If set to a non-empty string value, override the values of all the other
103419 internationalization variables.

103420 *LC_COLLATE*
103421 Determine the locale for character collation information in determining the
103422 pathname collation sequence.

103423 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
103424 characters (for example, single-byte as opposed to multi-byte characters in
103425 arguments) and which characters are defined as printable (character class **print**).

103426 *LC_MESSAGES*
103427 Determine the locale that should be used to affect the format and contents of
103428 diagnostic messages written to standard error.

103429 *LC_TIME* Determine the format and contents for date and time strings written by *ls*.

103430 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

103431 *TZ* Determine the timezone for date and time strings written by *ls*. If *TZ* is unset or
103432 null, an unspecified default timezone shall be used.

103433 **ASYNCHRONOUS EVENTS**

103434 Default.

103435 **STDOUT**

103436 The default format shall be to list one entry per line to standard output; the exceptions are to
103437 terminals or when one of the **–C**, **–m**, or **–x** options is specified. If the output is to a terminal, the
103438 format is implementation-defined.

103439 In the formats specified below, except where specified otherwise the <*pathname*> field shall

103440 consist of the file's pathname and, if the –**F** or –**p** option is specified, any indicator character
103441 written after the pathname as described for those options.

103442 When –**m** is specified, the format used for the last element of the list shall be:

103443 `"%s\n", <pathname>`

103444 The format used for each other element of the list shall be:

103445 `"%s,%s", <pathname>, <separator>`

103446 where, if there is not room for the next element of the list to fit within the current line length,
103447 <*separator*> is a string containing an optional <space> character and a mandatory <newline>
103448 character; otherwise it is a single <space> character.

103449 If the –**i** option is specified, the file's file serial number (see XBD **<sys/stat.h>**) shall be written in
103450 the following format before any other output for the corresponding entry:

103451 `%u ", <file serial number>`

103452 If the –**l** option is specified, the following information shall be written for files other than
103453 character special and block special files:

```
103454 "%s %u %s %s %u %s %s\n", <file mode>, <number of links>,
103455     <owner name>, <group name>, <size>, <date and time>,
103456     <pathname>
```

103457 If the –**l** option is specified, the following information shall be written for character special and
103458 block special files:

```
103459 "%s %u %s %s %s %s %s\n", <file mode>, <number of links>,
103460     <owner name>, <group name>, <device info>, <date and time>,
103461     <pathname>
```

103462 In both cases if the file is a symbolic link and the –**L** option is also specified, this information
103463 shall be for the file resolved from the symbolic link, except that the <*pathname*> field shall
103464 contain the pathname of the symbolic link itself. If the file is a symbolic link and the –**L** option is
103465 not specified, this information shall be about the link itself and the <*pathname*> field shall be one
103466 of the following forms:

103467 • `"%s%sΔ->Δ%s", <pathname of link>, <link type indicator>,`
103468 `<contents of link>`

103469 • `"%sΔ->Δ%s%s", <pathname of link>, <contents of link>,`
103470 `<file type indicator>`

103471 • `"%s%sΔ->Δ%s%s", <pathname of link>, <link type indicator>,`
103472 `<contents of link>, <file type indicator>`

103473 where <*link type indicator*> is a <commercial-at> (`'@'`) if the –**F** option is specified, or an empty
103474 string otherwise and <*file type indicator*> is the required indicator character, if any, for the file
103475 resolved from the symbolic link if the –**F** or –**p** option is specified, or an empty string otherwise.
103476 If pathname resolution fails when following the symbolic link, this shall not be treated as an
103477 error and the <*file type indicator*> field shall be an empty string.

103478 XSI The –**n**, –**g**, and –**o** options use the same format as –**l**, but with omitted items and their
103479 associated <blank> characters. See the OPTIONS section.

103480 In both the preceding –**l** forms, if <*owner name*> or <*group name*> cannot be determined, or if –**n**
103481 is given, they shall be replaced with their associated numeric values using the format `%u`.

The <*size*> field shall contain the value that would be returned for the file in the *st_size* field of **struct stat** (see XBD **<sys/stat.h>**). Note that for some file types this value is unspecified.

The <*device info*> field shall contain implementation-defined information associated with the device in question.

The <*date and time*> field shall contain the appropriate date and timestamp of when the file was last modified. In the POSIX locale, the field shall be the equivalent of the output of the following *date* command:

```
date "+%b %e %H:%M"
```

if the file has been modified in the last six months, or:

```
date "+%b %e %Y"
```

(where two <space> characters are used between `%e` and `%Y`) if the file has not been modified in the last six months or if the modification date is in the future, except that, in both cases, the final <newline> produced by *date* shall not be included and the output shall be as if the *date* command were executed at the time of the last modification date of the file rather than the current time. When the *LC_TIME* locale category is not set to the POSIX locale, a different format and order of presentation of this field may be used.

If the pathname was specified as a *file* operand, it shall be written as specified.

XSI The file mode written under the **–l**, **–n**, **–g**, and **–o** options shall consist of the following format:

```
"%c%s%s%s%s", <entry type>, <owner permissions>,
    <group permissions>, <other permissions>,
    <optional alternate access method flag>
```

The <*optional alternate access method flag*> shall be the empty string if there is no alternate or additional access control method associated with the file; otherwise, it shall be a string containing a single printable character that is not a <blank>.

The <*entry type*> character shall describe the type of file, as follows:

`d` Directory.

`b` Block special file.

`c` Character special file.

`l` (ell) Symbolic link.

`p` FIFO.

`s` Socket.

– Regular file.

Implementations may add other characters to this list to represent other implementation-defined file types.

The next three fields shall be three characters each:

<*owner permissions*>
Permissions for the file owner class (see XBD Section 4.7, on page 97).

<*group permissions*>
Permissions for the file group class.

103521 <*other permissions*>
103522 Permissions for the file other class.

103523 Each field shall have three character positions:

103524 1. If `'r'`, the file is readable; if `'-'`, the file is not readable.

103525 2. If `'w'`, the file is writable; if `'-'`, the file is not writable.

103526 3. The first of the following that applies:

103527 `S` If in <*owner permissions*>, the file is not executable and set-user-ID mode is set. If in
103528 <*group permissions*>, the file is not executable and set-group-ID mode is set.

103529 `s` If in <*owner permissions*>, the file is executable and set-user-ID mode is set. If in
103530 <*group permissions*>, the file is executable and set-group-ID mode is set.

103531 XSI `T` If in <*other permissions*> and the file is a directory, search permission is not granted to
103532 others, and the restricted deletion flag is set.

103533 XSI `t` If in <*other permissions*> and the file is a directory, search permission is granted to
103534 others, and the restricted deletion flag is set.

103535 `x` The file is executable or the directory is searchable.

103536 `-` None of the attributes of `'S'`, `'s'`, `'T'`, `'t'`, or `'x'` applies.

103537 Implementations may add other characters to this list for the third character position.
103538 Such additions shall, however, be written in lowercase if the file is executable or
103539 searchable, and in uppercase if it is not.

103540 XSI If any of the **–l**, **–n**, **–s**, **–g**, or **–o** options is specified, each list of files within a directory shall be
103541 preceded by a status line indicating the number of file system blocks occupied by the listed files
103542 for that directory in 512-byte units if the **–k** option is not specified, or 1 024-byte units if the **–k**
103543 option is specified, rounded up to the next integral number of units, if necessary. In the POSIX
103544 locale, the format shall be:

103545 `"total %u\n", <number of units in the directory>`

103546 If more than one directory, or a combination of non-directory files and directories are written,
103547 either as a result of specifying multiple operands, or the **–R** option, each list of files within a
103548 directory shall be preceded by:

103549 `"\n%s:\n", <directory name>`

103550 The above string may be omitted for the directory named by the operand if only one operand is
103551 present. It may also be omitted for dot (`'.'`) if no operands are present. If this string is the first
103552 thing to be written, the first <newline> shall not be written. This output shall precede the
103553 number of units in the directory.

103554 If the **–s** option is given, each file shall be written with the number of blocks used by the file.
103555 XSI Along with **–C**, **–1**, **–m**, or **–x**, the number and a <space> shall precede the filename; with **–l**, **–n**,
103556 **–g**, or **–o**, they shall precede each line describing a file.

103557 **STDERR**
103558 The standard error shall be used only for diagnostic messages.

103559 **OUTPUT FILES**
103560 None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

It is difficult for an application to use every part of the file modes field of *ls* **–l** in a portable manner. Certain file types and executable bits are not guaranteed to be exactly as shown, as implementations may have extensions. Applications can use this field to pass directly to a user printout or prompt, but actions based on its contents should generally be deferred, instead, to the *test* utility.

The output of *ls* (with the **–l** and related options) contains information that logically could be used by utilities such as *chmod* and *touch* to restore files to a known state. However, this information is presented in a format that cannot be used directly by those utilities or be easily translated into a format that can be used. A character has been added to the end of the permissions string so that applications at least have an indication that they may be working in an area they do not understand instead of assuming that they can translate the permissions string into something that can be used. Future versions or related documents may define one or more specific characters to be used based on different standard additional or alternative access control mechanisms.

As with many of the utilities that deal with filenames, the output of *ls* for multiple files or in one of the long listing formats must be used carefully on systems where filenames can contain embedded white space. Systems and system administrators should institute policies and user training to limit the use of such filenames.

The number of disk blocks occupied by the file that it reports varies depending on underlying file system type, block size units reported, and the method of calculating the number of blocks. On some file system types, the number is the actual number of blocks occupied by the file (counting indirect blocks and ignoring holes in the file); on others it is calculated based on the file size (usually making an allowance for indirect blocks, but ignoring holes).

The `total` number provided when using *ls* **–l** does not necessarily correspond to the space that would be reclaimed if all the listed files were removed, because of hard links (and symbolic links if **–L** is present). The space for each listed file is counted in the total regardless of any relationship between the files.

EXAMPLES

An example of a small directory tree being fully listed with *ls* **–laRF a** in the POSIX locale:

```
a:
total 11
drwxr-xr-x   3 fox      prog          64 Jul  4 12:07 ./
drwxrwxrwx   4 fox      prog        3264 Jul  4 12:09 ../
drwxr-xr-x   2 fox      prog          48 Jul  4 12:07 b/
-rwxr--r--   1 fox      prog         572 Jul  4 12:07 foo*

a/b:
total 4
```



```
drwxr-xr-x   2 fox      prog          48 Jul  4 12:07 ./
drwxr-xr-x   3 fox      prog          64 Jul  4 12:07 ../
-rw-r--r--   1 fox      prog         700 Jul  4 12:07 bar
```

where the `"a:"` line may be omitted by some implementations.

RATIONALE

Some historical implementations of the *ls* utility show all entries in a directory except dot and dot-dot when a superuser invokes *ls* without specifying the **–a** option. When ``normal'' users invoke *ls* without specifying **–a**, they should not see information about any files with names beginning with a <period> unless they were named as *file* operands.

Implementations are expected to traverse arbitrary depths when processing the **–R** option. The only limitation on depth should be based on running out of physical storage for keeping track of untraversed directories.

The **–1** (one) option was historically found in BSD and BSD-derived implementations only. It is required in this volume of POSIX.1-2024 so that conforming applications might ensure that output is one entry per line, even if the output is to a terminal.

The **–S** option was added in Issue 7, but had been provided by several implementations for many years. The description given in the standard documents historic practice, but does not match much of the documentation that described its behavior. Historical documentation typically described it as something like:

> **–S** Sort by size (largest size first) instead of by name. Special character devices (listed last) are sorted by name.

even though the file type was never considered when sorting the output. Character special files do typically sort close to the end of the list because their file size on most implementations is zero. But they are sorted alphabetically with any other files that happen to have the same file size (zero), not sorted separately and added to the end.

This volume of POSIX.1-2024 is frequently silent about what happens when mutually-exclusive options are specified. Except for **–R**, **–d**, and **–f**, the *ls* utility is required to accept multiple options from each mutually-exclusive option set without treating them as errors and to use the behavior specified by the last option given in each mutually-exclusive set. Since *ls* is one of the most aliased commands, it is important that the implementation perform intuitively. For example, if the alias were:

```
alias ls="ls -C"
```

and the user typed *ls* **–1** (one), single-text-column output should result, not an error.

The **–g**, **–l** (ell), **–n**, and **–o** options are not mutually-exclusive options. They all enable long format output. They work together to determine whether the file's owner is written (no if **–g** is present), file's group is written (no if **–o** is present), and if the file's group or owner is written whether it is written as the name (default) or a string representation of the UID or GID number (if **–n** is present). The **–C**, **–m**, **–x**, and **–1** (one) are mutually-exclusive options and the first three of these disable long format output. The **–1** (one) option does not directly change whether or not long format output is enabled, but by overriding **–C**, **–m**, and **–x**, it can re-enable long format output that had been disabled by one of these options.

Earlier versions of this standard did not describe the BSD **–A** option (like **–a**, but dot and dot-dot are not written out). It has been added due to widespread implementation.

Implementations may make **–q** the default for terminals to prevent trojan horse attacks on terminals with special escape sequences. This is not required because:

- Some control characters may be useful on some terminals; for example, a system might write them as `"\001"` or `"^A"`.
- Special behavior for terminals is not relevant to applications portability.

An early proposal specified that the <*optional alternate access method flag*> had to be `'+'` if there was an alternate access method used on the file or <space> if there was not. This was changed to be <space> if there is not and a single printable character if there is. This was done for three reasons:

1. There are historical implementations using characters other than `'+'`.
2. There are implementations that vary this character used in that position to distinguish between various alternate access methods in use.
3. The standard developers did not want to preclude future specifications that might need a way to specify more than one alternate access method.

Nonetheless, implementations providing a single alternate access method are encouraged to use `'+'`.

Earlier versions of this standard did not have the **–k** option, which meant that the **–s** option could not be used portably as its block size was implementation-defined, and the units used to specify the number of blocks occupied by files in a directory in an *ls* **–l** listing were fixed as 512-byte units. The **–k** option has been added to provide a way for the **–s** option to be used portably, and for consistency it also changes the aforementioned units from 512-byte to 1 024-byte.

The <*date and time*> field in the **–l** format is specified only for the POSIX locale. As noted, the format can be different in other locales. No mechanism for defining this is present in this volume of POSIX.1-2024, as the appropriate vehicle is a messaging system; that is, the format should be specified as a ``message''.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

Allowing **–f** to ignore the **–A**, **–g**, **–l**, **–n**, **–o**, and **–s** options may be removed in a future version.

SEE ALSO

chmod, *find*, *readlink*

XBD Section 7.3.2 (on page 139), Section 4.7 (on page 97), Chapter 8 (on page 167), Section 12.2 (on page 215), **<sys/stat.h>**

XSH *fstatat*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

A second FUTURE DIRECTION is added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the –**F** option, other symbols are allowed for other file types.

Treatment of symbolic links is added, as defined in the IEEE P1003.2b draft standard.

The Open Group Base Resolution bwg2001-010 is applied, adding the `T` and `t` fields as part of the XSI option.

Issue 7

Austin Group Interpretation 1003.1-2001 #101 is applied, clarifying the optional alternate access method flag in the STDOUT section.

Austin Group Interpretation 1003.1-2001 #128 is applied, clarifying the DESCRIPTION and the definition of the –**R** option.

Austin Group Interpretation 1003.1-2001 #129 is applied, clarifying the behavior of *ls* when no operands are specified.

Austin Group Interpretation 1003.1-2001 #198 is applied, clarifying the requirements for the –**H** option.

SD5-XCU-ERN-50 is applied, adding the –**A** option.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The –**S** option is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The –**f**, –**m**, –**n**, –**p**, –**s**, and –**x** options are moved from the XSI option to the Base.

The description of the –**f**, –**s**, and –**t** options are revised and the –**k** option is added.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0098 [424], XCU/TC1-2008/0099 [424], XCU/TC1-2008/0100 [424], XCU/TC1-2008/0101 [424], XCU/TC1-2008/0102 [424], XCU/TC1-2008/0103 [424], XCU/TC1-2008/0104 [424], XCU/TC1-2008/0105 [423,424], XCU/TC1-2008/0106 [424], XCU/TC1-2008/0107 [424], XCU/TC1-2008/0108 [424], XCU/TC1-2008/0109 [424], XCU/TC1-2008/0110 [424], XCU/TC1-2008/0111 [423], XCU/TC1-2008/0112 [117], XCU/TC1-2008/0113 [117], XCU/TC1-2008/0114 [117], XCU/TC1-2008/0115 [424], and XCU/TC1-2008/0116 [424] are applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0115 [963] and XCU/TC2-2008/0116 [963] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1023 is applied, clarifying the –**R** option with respect to subdirectory filenames beginning with a <period>.

Austin Group Defect 1070 is applied, requiring that any filenames or pathnames that collate equally are further compared byte-by-byte using the collating sequence for the POSIX locale.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1146 is applied, clarifying the requirements for the status line written by *ls* –**l**.

Austin Group Defect 1147 is applied, clarifying the requirements for trailing file type indicators (such as `'/'` for a directory).

Austin Group Defect 1148 is applied, clarifying the behavior of the **−S** option for symbolic links.

Austin Group Defect 1149 is applied, inserting a comma in the description of the **−r** option.

Austin Group Defect 1185 is applied, changing the *COLUMNS* entry in ENVIRONMENT VARIABLES.

Austin Group Defect 1217 is applied, adding file type indicators for sockets.

Austin Group Defect 1261 is applied, changing the STDOUT and EXAMPLES sections in relation to the <*directory name*> output.

Austin Group Defect 1457 is applied, adding *readlink* to the SEE ALSO section.

Austin Group Defect 1703 is applied, changing ``at-sign'' to ``<commercial-at>''.

NAME

m4 — macro processor

SYNOPSIS

```
m4 [-s] [-D name[=val]]... [-U name]... [file...]
```

DESCRIPTION

The *m4* utility is a macro processor that shall read one or more text files, process them according to their included macro statements, and write the results to standard output.

OPTIONS

The *m4* utility shall conform to XBD Section 12.2 (on page 215), except that the order of the **–D** and **–U** options shall be significant, and options can be interspersed with operands.

The following options shall be supported:

–s Enable line synchronization output for the *c17* preprocessor phase (that is, **#line** directives).

–D *name*[=*val*]
Define *name* to *val* or to null if =*val* is omitted.

–U *name* Undefine *name*.

OPERANDS

The following operand shall be supported:

file A pathname of a text file to be processed. If no *file* is given, or if it is `'-'`, the standard input shall be read.

STDIN

The standard input shall be a text file that is used if no *file* operand is given, or if it is `'-'`.

INPUT FILES

The input file named by the *file* operand shall be a text file.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *m4*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall be the same as the input files, after being processed for macro expansion.

STDERR

The standard error shall be used to display strings with the **errprint** macro, macro tracing enabled by the **traceon** macro, the defined text for macros written by the **dumpdef** macro, or for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

The *m4* utility shall compare each token from the input against the set of built-in and user-defined macros. If the token matches the name of a macro, then the token shall be replaced by the macro's defining text, if any, then scanning for tokens shall resume at the start of the macro's defining text concatenated with the subsequent input. If a token does not match the name of a macro, it shall be written to standard output. Macros may have arguments, in which case the arguments shall be substituted into the defining text before it is rescanned.

No special meaning shall be given to characters enclosed between matching left and right quoting strings, other than identifying nested quoting while finding the matching right quoting string, but the outermost quoting strings shall themselves be discarded. By default, the left quoting string consists of a grave accent (backquote) and the right quoting string consists of an acute accent (single-quote); see also the **changequote** macro.

Comments are written but not scanned for matching macro names; by default, the begin-comment string consists of the <number-sign> character and the end-comment string consists of a <newline>. See also the **changecom** and **dnl** macros.

Name tokens shall consist of the longest possible sequence of letters, digits, and underscores, where the first character is not a digit. Tokens not of this form shall not be treated as name tokens. A macro call is a name token that matches the name of a built-in or user-defined macro. Macro calls can have either of the following forms, which shall be distinguished by whether or not the macro name is immediately followed by a <left-parenthesis>:

name

name(*arg1*, *arg2*, ..., *argn*)

The application shall ensure that the <left-parenthesis> immediately follows the name of the macro. If a token matching the name of a macro is not followed by a <left-parenthesis>, it shall be handled as a use of that macro without arguments.

If a macro name is followed by a <left-parenthesis>, the subsequent text shall be tokenized and expanded until a token is encountered that is not a quoted string and whose expansion includes a matching unquoted <right-parenthesis>. The expanded text between the <left-parenthesis> and the matching unquoted <right-parenthesis> is the macro's argument text. An unquoted <comma> character within the macro's argument text shall mark the end of one argument and the beginning of the next argument unless the unquoted <comma> is enclosed within a nested unquoted <left-parenthesis>, <right-parenthesis> pair. The unquoted <comma> characters that separate the arguments, and any unquoted white-space characters at the beginning of each argument, shall be discarded. All other characters in the macro's argument text, including any white-space characters at the end of an argument and any nested parenthesized text, shall be retained. The input text containing the macro name, the following <left-parenthesis>, and all tokens up to and including the token whose expansion contained the matching unquoted <right-parenthesis> shall be replaced, and tokenization shall resume on the result of performing

103832 argument substition on the macro's defining text followed by any expanded text that followed
103833 the matching unquoted <right-parenthesis>. Otherwise, the macro name was not followed by a
103834 <left-parenthesis>, and tokenization shall resume on the result of performing argument
103835 substitution with zero arguments on the macro's defining text.

103836 During argument substitution, arguments shall be positionally defined and referenced. The
103837 string `"$1"` in the defining text shall be replaced by the first argument. Systems shall support at
103838 least nine arguments; only the first nine can be referenced, using the strings `"$1"` to `"$9"`,
103839 inclusive. The string `"$0"` shall be replaced with the name of the macro. The string `"$#"` shall
103840 be replaced by the number of arguments as a minimal string of decimal digits (`'0'` if the macro
103841 was invoked without being followed by a <left-parenthesis>, otherwise 1 more than the number
103842 of unquoted <comma> characters that divided arguments in the macro's argument text). The
103843 string `"$*"` shall be replaced by a list of all of the arguments, separated by <comma> characters.
103844 The string `"$@"` shall be replaced by a list of all of the arguments separated by <comma>
103845 characters, and each argument shall be quoted using the current left and right quoting strings.
103846 The string `"${"` produces unspecified behavior.

103847 If fewer arguments are supplied than are in the macro definition, the omitted arguments are
103848 taken to be null. It is not an error if more arguments are supplied than are in the macro
103849 definition.

103850 The *m4* utility shall make available the following built-in macros. They can be redefined, but
103851 once this is done the original meaning is lost. Their values shall be null unless otherwise stated.
103852 In the descriptions below, the term *defining text* refers to the value of the macro: the second
103853 argument to the **define** macro, among other things. Except for the first argument to the **eval**
103854 macro, all numeric arguments to built-in macros shall be interpreted as decimal values. The
103855 string values produced as the defining text of the **decr**, **divnum**, **incr**, **index**, **len**, and **sysval**
103856 built-in macros shall be in the form of a decimal-constant as defined in the C language.

103857 **changecom** The **changecom** macro shall set the begin-comment and end-comment strings.
103858 With no arguments, the comment mechanism shall be disabled. With a single non-
103859 null argument, that argument shall become the begin-comment and the <newline>
103860 shall become the end-comment string. With two non-null arguments, the first
103861 argument shall become the begin-comment string and the second argument shall
103862 become the end-comment string. The behavior is unspecified if either argument is
103863 provided but null, or if either argument includes letters, digits, underscore, or
103864 <left-parenthesis>. Systems shall support comment strings of at least five
103865 characters.

103866 **changequote** The **changequote** macro shall set the begin-quote and end-quote strings. With no
103867 arguments, the quote strings shall be set to the default values (that is, `` `' ``). The
103868 behavior is unspecified if there is a single argument, or if either argument is null or
103869 includes letters, digits, underscore, or <left-parenthesis>. With two non-null
103870 arguments, the first argument shall become the begin-quote string and the second
103871 argument shall become the end-quote string. Systems shall support quote strings
103872 of at least five characters.

103873 **decr** The defining text of the **decr** macro shall be its first argument decremented by 1. It
103874 shall be an error to specify an argument containing any non-numeric characters.
103875 The behavior is unspecified if **decr** is not immediately followed by a <left-
103876 parenthesis>.

103877 **define** The second argument shall become the defining text of the macro whose name is
103878 the first argument. It is unspecified whether the **define** macro deletes all prior
103879 definitions of the macro named by its first argument or preserves all but the

current definition of the macro. The behavior is unspecified if **define** is not immediately followed by a <left-parenthesis>.

defn The defining text of the **defn** macro shall be the quoted definition (using the current quoting strings) of its arguments. The behavior is unspecified if **defn** is not immediately followed by a <left-parenthesis>.

divert The *m4* utility maintains nine temporary buffers, numbered 1 to 9, inclusive. When the last of the input has been processed, any output that has been placed in these buffers shall be written to standard output in buffer-numerical order. The **divert** macro shall divert future output to the buffer specified by its argument. Specifying no argument or an argument of 0 shall resume the normal output process. Output diverted to a stream with a negative number shall be discarded. Behavior is implementation-defined if a stream number larger than 9 is specified. It shall be an error to specify an argument containing any non-numeric characters.

divnum The defining text of the **divnum** macro shall be the number of the current output stream as a string.

dnl The **dnl** macro shall cause *m4* to discard all input characters up to and including the next <newline>.

dumpdef The **dumpdef** macro shall write the defined text to standard error for each of the macros specified as arguments, or, if no arguments are specified, for all macros.

errprint The **errprint** macro shall write its arguments to standard error. The behavior is unspecified if **errprint** is not immediately followed by a <left-parenthesis>.

eval The **eval** macro shall evaluate its first argument as an arithmetic expression, using signed integer arithmetic with at least 32-bit precision. At least the following C-language operators shall be supported, with precedence, associativity, and behavior as described in Section 1.1.2.1 (on page 2457):

```
()
unary +
unary -
~
!
binary *
/
%
binary +
binary -
<<
>>
<
<=
>
>=
==
!=
binary &
^
|
&&
```


103927 ||

103928 Systems shall support octal and hexadecimal numbers as in the ISO C standard.
103929 The second argument, if specified, shall set the radix for the result; if the argument
103930 is blank or unspecified, the default is 10. Behavior is unspecified if the radix falls
103931 outside the range 2 to 36, inclusive. The third argument, if specified, sets the
103932 minimum number of digits in the result. Behavior is unspecified if the third
103933 argument is less than zero. It shall be an error to specify the second or third
103934 argument containing any non-numeric characters. The behavior is unspecified if
103935 **eval** is not immediately followed by a <left-parenthesis>.

103936 **ifdef** If the first argument to the **ifdef** macro is defined, the defining text shall be the
103937 second argument. Otherwise, the defining text shall be the third argument, if
103938 specified, or the null string, if not. The behavior is unspecified if **ifdef** is not
103939 immediately followed by a <left-parenthesis>.

103940 **ifelse** The **ifelse** macro takes three or more arguments. If the first two arguments
103941 compare as equal strings, the defining text shall be the third argument. If the first
103942 two arguments do not compare as equal strings and there are three arguments, the
103943 defining text shall be null. If the first two arguments do not compare as equal
103944 strings and there are four or five arguments, the defining text shall be the fourth
103945 argument. If the first two arguments do not compare as equal strings and there are
103946 six or more arguments, the first three arguments shall be discarded and processing
103947 shall restart with the remaining arguments. The behavior is unspecified if **ifelse** is
103948 not immediately followed by a <left-parenthesis>.

103949 **include** The defining text for the **include** macro shall be the contents of the file named by
103950 the first argument. It shall be an error if the file cannot be read. The behavior is
103951 unspecified if **include** is not immediately followed by a <left-parenthesis>.

103952 **incr** The defining text of the **incr** macro shall be its first argument incremented by 1. It
103953 shall be an error to specify an argument containing any non-numeric characters.
103954 The behavior is unspecified if **incr** is not immediately followed by a <left-
103955 parenthesis>.

103956 **index** The defining text of the **index** macro shall be the first character position (as a
103957 string) in the first argument where a string matching the second argument begins
103958 (zero origin), or –1 if the second argument does not occur. The behavior is
103959 unspecified if **index** is not immediately followed by a <left-parenthesis>.

103960 **len** The defining text of the **len** macro shall be the length (as a string) of the first
103961 argument. The behavior is unspecified if **len** is not immediately followed by a
103962 <left-parenthesis>.

103963 **m4exit** Exit from the *m4* utility. If the first argument is specified, it shall be the exit code. If
103964 no argument is specified, the exit code shall be zero. It shall be an error to specify
103965 an argument containing any non-numeric characters. If the first argument is zero
103966 or no argument is specified, and an error has previously occurred (for example, a
103967 *file* operand that could not be opened), the exit status shall be non-zero.

103968 **m4wrap** The first argument shall be processed when EOF is reached. If the **m4wrap** macro
103969 is used multiple times, the arguments specified shall be processed in the order in
103970 which the **m4wrap** macros were processed. The behavior is unspecified if **m4wrap**
103971 is not immediately followed by a <left-parenthesis>.

mkstemp The defining text shall be as if it were the resulting pathname after a successful call to the *mkstemp*( ) function defined in the System Interfaces volume of POSIX.1-2024 called with the first argument to the macro invocation. If a file is created, that file shall be closed. If a file could not be created, the *m4* utility shall write a diagnostic message to standard error and shall continue processing input but its final exit status shall be non-zero; the defining text of the macro shall be the empty string. The behavior is unspecified if **mkstemp** is not immediately followed by a <left-parenthesis>.

popdef The **popdef** macro shall delete the current definition of its arguments, replacing that definition with the previous one. If there is no previous definition, the macro is undefined. The behavior is unspecified if **popdef** is not immediately followed by a <left-parenthesis>.

pushdef The **pushdef** macro shall be equivalent to the **define** macro with the exception that it shall preserve any current definition for future retrieval using the **popdef** macro. The behavior is unspecified if **pushdef** is not immediately followed by a <left-parenthesis>.

shift The defining text for the **shift** macro shall be a comma-separated list of its arguments except the first one. Each argument shall be quoted using the current quoting strings. The behavior is unspecified if **shift** is not immediately followed by a <left-parenthesis>.

sinclude The **sinclude** macro shall be equivalent to the **include** macro, except that it shall not be an error if the file is inaccessible. The behavior is unspecified if **sinclude** is not immediately followed by a <left-parenthesis>.

substr The defining text for the **substr** macro shall be the substring of the first argument beginning at the zero-offset character position specified by the second argument. The third argument, if specified, shall be the number of characters to select; if not specified, the characters from the starting point to the end of the first argument shall become the defining text. It shall not be an error to specify a starting point beyond the end of the first argument and the defining text shall be null. It shall be an error to specify an argument containing any non-numeric characters. The behavior is unspecified if **substr** is not immediately followed by a <left-parenthesis>.

syscmd The **syscmd** macro shall interpret its first argument as a shell command line. The defining text shall be the string result of that command. The string result shall not be rescanned for macros while setting the defining text. No output redirection shall be performed by the *m4* utility. The exit status value from the command can be retrieved using the **sysval** macro. The behavior is unspecified if **syscmd** is not immediately followed by a <left-parenthesis>.

sysval The defining text of the **sysval** macro shall be the exit value of the utility last invoked by the **syscmd** macro (as a string).

traceon The **traceon** macro shall enable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros. The trace output shall be written to standard error in an unspecified format.

traceoff The **traceoff** macro shall disable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros.

translit The defining text of the **translit** macro shall be the first argument with every character that occurs in the second argument replaced with the corresponding character from the third argument. If no replacement character is specified for some source character because the second argument is longer than the third argument, that character shall be deleted from the first argument in **translit**'s defining text. The behavior is unspecified if the `'-'` character appears within the second or third argument anywhere besides the first or last character. The behavior is unspecified if the same character appears more than once in the second argument. The behavior is unspecified if **translit** is not immediately followed by a <left-parenthesis>.

undefine The **undefine** macro shall delete all definitions (including those preserved using the **pushdef** macro) of the macros named by its arguments. The behavior is unspecified if **undefine** is not immediately followed by a <left-parenthesis>.

undivert The **undivert** macro shall cause immediate output of any text in temporary buffers named as arguments, or all temporary buffers if no arguments are specified. Buffers can be undiverted into other temporary buffers. Undiverting shall discard the contents of the temporary buffer. The behavior is unspecified if an argument contains any non-numeric characters.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred

If the **m4exit** macro is used, the exit value can be specified by the input file.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The **defn** macro is useful for renaming macros, especially built-ins.

Since **eval** defers to the ISO C standard, some operations have undefined behavior. In some implementations, division or remainder by zero cause a fatal signal, even if the division occurs on the short-circuited branch of `"&&"` or `"||"`. Any operation that overflows in signed arithmetic produces undefined behavior. Likewise, using the **shift** operators with a shift amount that is not positive and smaller than the precision is undefined, as is shifting a negative number to the right. Historically, not all implementations obeyed C-language precedence rules: `'~'` and `'!'` were lower than `'=='`; `'=='` and `'!='` were not lower than `'<'`; and `'|'` was not lower than `'^'`; the liberal use of `"()"` can force the desired precedence even with these non-compliant implementations. Furthermore, some traditional implementations treated `'^'` as an exponentiation operator, although most implementations now use `"**"` as an extension for this purpose.

When a macro has been multiply defined via the **pushdef** macro, it is unspecified whether the **define** macro will alter only the most recent definition (as though by **popdef** and **pushdef**), or replace the entire stack of definitions with a single definition (as though by **undefine** and **pushdef**). An application desiring particular behavior for the **define** macro in this case can redefine it accordingly.

EXAMPLES

If the file **m4src** contains the lines:

```
The value of `VER' is "VER".
ifdef(`VER', ``VER'' is defined to be VER., VER is not defined.)
ifelse(VER, 1, ``VER'' is `VER'.)
ifelse(VER, 2, ``VER'' is `VER'., ``VER'' is not 2.)
end
```

then the command

```
m4 m4src
```

or the command:

```
m4 -U VER m4src
```

produces the output:

```
The value of VER is "VER".
VER is not defined.

VER is not 2.
end
```

The command:

```
m4 -D VER m4src
```

produces the output:

```
The value of VER is "".
VER is defined to be .

VER is not 2.
end
```

The command:

```
m4 -D VER=1 m4src
```

produces the output:

```
The value of VER is "1".
VER is defined to be 1.
VER is 1.
VER is not 2.
end
```

The command:

```
m4 -D VER=2 m4src
```

produces the output:

```
The value of VER is "2".
VER is defined to be 2.

VER is 2.
end
```

In the following six examples, an additional line is evaluated after this prologue of three definitions:


```
define(`macro', `argument 2 is :`$2':, called with $# arguments')dnl
define(`argumentsa', `Arguments')dnl
define(`a', `.')dnl
```

1. The additional line:

   ```
   macro`'a
   ```

   produces:

   ```
   argument 2 is ::, called with 0 arguments.
   ```

   Explanation: *macro* is called with 0 arguments (as shown by the `$#` substitution), the substitution of `$2` is the empty string, and the empty quoted string after the expansion text prevents concatenation with the subsequent `'a'`, which in turn lets macro *a* expand to the final `'.'`.

2. The additional line:

   ```
   macro()a
   ```

   produces:

   ```
   argument 2 is ::, called with 1 Arguments
   ```

   Explanation: *macro* is called with one (empty string) argument; then the defining text ending in `"arguments"` is concatenated with the subsequent `'a'` to form the next macro name *argumentsa* which is expanded into `Arguments` before the final output.

3. The additional line:

   ```
   macro(  1, ( ,2,) ,  `3')
   ```

   produces:

   ```
   argument 2 is :( ,2,) :, called with 3 arguments
   ```

   Explanation: Leading (but not internal or trailing) space is removed before the argument substituted for `$2`, and the unquoted commas embedded in parentheses do not delineate arguments.

4. The additional line:

   ```
   macro(  `1', `mac2(,`2',)', `3')
   ```

   produces:

   ```
   argument 2 is :mac2(,`2',):, called with 3 arguments
   ```

   Explanation: Regardless of whether *mac2* is a defined macro, quoting in the macro call prevents interpretation of `"mac2"` during argument collection, and the quoting in the defining text of *macro* prevents interpretation of `"mac2"` in the substitution of `$2` during rescan of the output of *macro*.

5. The additional line:

   ```
   undefine(`mac2')macro(  1, mac2(,2,), 3)
   ```

   produces:

   ```
   argument 2 is :mac2(,2,):, called with 3 arguments
   ```

   Explanation: *mac2* is not a macro name when scanned during argument collection, so it and the subsequent parenthesized text are used literally.

6. The additional line:

   ```
   define(`mac2', `hi $@')macro(  1, mac2( ,2,), 3)
   ```

   produces:

   ```
   argument 2 is :hi :, called with 5 arguments
   ```

   Explanation: *mac2* is a macro name, so collecting the arguments to *macro* requires scanning the output of *mac2(,2,)* (the text `hi `',`2',`'` after substitution of `$@`); this output contains unquoted commas causing additional arguments to be visible to *macro*.

RATIONALE

Historic System V-based behavior treated `"${"` in a macro definition as two literal characters. However, this sequence is left unspecified so that implementations may offer extensions such as `"${11}"` meaning the eleventh argument. Macros can still be defined with appropriate uses of nested quoting to result in a literal `"${"` in the output after rescanning removes the nested quotes.

In the **translit** built-in, historic System V-based behavior treated `'-'` as a literal; GNU behavior treats it as a range. This version of the standard allows either behavior.

Earlier versions of this standard allowed the exit status to be either zero or non-zero when `m4exit(0)` is called after an error has occurred. Exiting with zero status is now disallowed as this hides the fact that an error occurred from shell scripts that check the exit status of *m4*.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

c17

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 5

The phrase ``the defined text for macros written by the **dumpdef** macro'' is added to the description of STDERR, and the description of **dumpdef** is updated to indicate that output is written to standard error. The description of **eval** is updated to indicate that the list of excluded C operators excludes unary `'&'` and `'.'`. In the description of **ifdef**, the phrase ``and it is not defined to be zero'' is deleted.

Issue 6

In the EXTENDED DESCRIPTION, the **eval** text is updated to include a `'&'` character in the excepted list.

The EXTENDED DESCRIPTION of **divert** is updated to clarify that there are only nine diversion buffers.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

The Open Group Base Resolution bwg2000-006 is applied.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/31 is applied, replacing the EXAMPLES section.

Issue 7

Austin Group Interpretation 1003.1-2001 #117 is applied, marking the **maketemp** macro obsolescent and adding a new **mkstemp** macro.

Austin Group Interpretation 1003.1-2001 #207 is applied, clarifying the handling of white-space characters that precede or trail any macro arguments.

SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands).

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

SD5-XCU-ERN-99 is applied, clarifying the definition of the **divert** macro in the EXTENDED DESCRIPTION.

SD5-XCU-ERN-100 is applied, clarifying the definition of the **syscmd** macro in the EXTENDED DESCRIPTION.

SD5-XCU-ERN-101 is applied, clarifying the definition of the **undivert** macro in the EXTENDED DESCRIPTION.

SD5-XCU-ERN-111 is applied to the EXTENDED DESCRIPTION, clarifying that the string `"${"` produces unspecified behavior.

SD5-XCU-ERN-112 is applied, updating the **changequote** macro.

SD5-XCU-ERN-118 is applied, clarifying the definition of the **define** macro in the EXTENDED DESCRIPTION and APPLICATION USAGE sections.

SD5-XCU-ERN-119 is applied, clarifying the definition of the **translit** macro in the EXTENDED DESCRIPTION and RATIONALE sections.

SD5-XCU-ERN-130, SD5-XCU-ERN-131, and SD5-XCU-ERN-137 are applied.

The *m4* utility is moved from the XSI option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0117 [241], XCU/TC1-2008/0118 [242,431], XCU/TC1-2008/0119 [242,431], and XCU/TC1-2008/0120 [325,430] are applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0117 [964], XCU/TC2-2008/0118 [970], and XCU/TC2-2008/0119 [964] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 984 is applied, requiring that the exit status of *m4* is non-zero when **m4exit** is called with a first argument of zero or with no arguments after an error has occurred.

Austin Group Defect 1072 is applied, clarifying the handling of macro arguments.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1330 is applied, removing the obsolescent **maketemp** macro.

Austin Group Defect 1514 is applied, changing the RATIONALE section to use the same terminology as the normative text to which it refers.

Austin Group Defect 1570 is applied, removing extra spacing in `"=="`.

Austin Group Defect 1658 is applied, changing ``whitespace characters'' to ``white-space

104223 characters''.

104224 Austin Group Defect 1730 is applied, adding square brackets around *file*... in the
104225 SYNOPSIS.

104226 **NAME**

104227 mailx — process messages

104228 **SYNOPSIS**

104229 **Send Mode**

104230 `mailx [-E] [-s subject] address...`

104231 **Receive Mode**

104232 UP `mailx -e`

104233 `mailx [-HiNn] [-F] [-u user]`

104234 `mailx -f [-HiNn] [-F] [file]`

104235 **DESCRIPTION**

104236 The *mailx* utility provides a message sending and receiving facility. It has two major modes,
104237 selected by the options used: Send Mode and Receive Mode.

104238 On systems that do not support the User Portability Utilities option, an application using *mailx*
104239 shall have the ability to send messages in an unspecified manner (Send Mode). Unless the first
104240 character of one or more lines is <tilde> (`'~'`), all characters in the input message shall appear in
104241 the delivered message, but additional characters may be inserted in the message before it is
104242 retrieved.

104243 UP On systems supporting the User Portability Utilities option, mail-receiving capabilities and other
104244 interactive features, Receive Mode, described below, also shall be enabled.

104245 **Send Mode**

104246 Send Mode can be used by applications or users to send messages from the text in standard
104247 input. The message shall be passed to the mail delivery software. The mail delivery software
104248 shall process passed messages according to the rules of IETF RFC 5322.

104249 UP **Receive Mode**

104250 Receive Mode is more oriented towards interactive users. Mail can be read and sent in this
104251 interactive mode.

104252 When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to
104253 messages. When sending mail, *mailx* allows editing, reviewing, and other modification of the
104254 message as it is entered.

104255 Incoming mail shall be stored in one or more unspecified locations for each user, collectively
104256 called the system *mailbox* for that user. When *mailx* is invoked in Receive Mode, the system
104257 mailbox shall be the default place to find new mail. As messages are read, they shall be marked
104258 to be moved to a secondary file for storage, unless specific action is taken. This secondary file is
104259 called the **mbox** and is normally located in the directory referred to by the *HOME* environment
104260 variable (see *MBOX* in the ENVIRONMENT VARIABLES section for a description of this file).
104261 Messages shall remain in this file until explicitly removed. When the **–f** option is used to read
104262 mail messages from secondary files, messages shall be retained in those files unless specifically
104263 removed. All three of these locations—system mailbox, **mbox**, and secondary file—are referred
104264 to in this section as simply ``mailboxes'', unless more specific identification is required.

OPTIONS

The *mailx* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported. (Only the **−E** and **−s** *subject* options are required on all systems. The other options are required only on systems supporting the User Portability Utilities option.)

−E Discard messages with an empty message body.

UP **−e** Test for the presence of mail in the system mailbox. The *mailx* utility shall write nothing and exit with a successful return code if there is mail to read.

UP **−f** Read messages from the file named by the *file* operand instead of the system mailbox. (See also **folder**.) If no *file* operand is specified, read messages from **mbox** instead of the system mailbox.

UP **−F** Record the message in a file named after the first recipient. The name is the login-name portion of the address found first in the **To** field in the message header. Overrides the **record** variable, if set (see Internal Variables in mailx, on page 3109).

UP **−H** Write a header summary only.

UP **−i** Ignore interrupts. (See also **ignore**.)

UP **−n** Do not initialize from the system default start-up file. See the EXTENDED DESCRIPTION section.

UP **−N** Do not write an initial header summary.

−s *subject* Set the **Subject** header field to *subject*. All characters in the *subject* string shall appear in the delivered message. The results are unspecified if *subject* is longer than {LINE_MAX} − 10 bytes or contains a <newline>.

UP **−u** *user* Read the system mailbox of the login name *user*. This shall only be successful if the invoking user has appropriate privileges to read the system mailbox of that user.

OPERANDS

The following operands shall be supported:

address Addressee of message. When **−n** is specified and no user start-up files are accessed (see the EXTENDED DESCRIPTION section), the user or application shall ensure this is an address to pass to the mail delivery system. Any system or user start-up files may enable aliases (see **alias** under Commands in mailx, on page 3112) that may modify the form of *address* before it is passed to the mail delivery system.

UP *file* A pathname of a file to be read instead of the system mailbox when **−f** is specified. The meaning of the *file* operand shall be affected by the contents of the **folder** internal variable; see Internal Variables in mailx (on page 3109).

STDIN

When *mailx* is invoked in Send Mode (the first synopsis line), standard input shall be the
UP message to be delivered to the specified addresses. When in Receive Mode, user commands shall be accepted from *stdin*. If the User Portability Utilities option is not supported, standard input lines beginning with a <tilde> (**'~'**) character produce unspecified results.

UP If the User Portability Utilities option is supported, then in both Send and Receive Modes, standard input lines beginning with the escape character (usually <tilde> (**'~'**)) shall affect processing as described in Command Escapes in mailx (on page 3122).

INPUT FILES

When *mailx* is used as described by this volume of POSIX.1-2024, the *file* operand (see the **−f** option) and the **mbox** shall be text files containing mail messages, formatted as described in the OUTPUT FILES section. The nature of the system mailbox is unspecified; it need not be a file.

ENVIRONMENT VARIABLES

UP Some of the functionality described in this section shall be provided on implementations that support the User Portability Utilities option as described in the text, and is not further shaded for this option.

The following environment variables shall affect the execution of *mailx*:

DEAD Determine the pathname of the file in which to save partial messages in case of interrupts or delivery errors. The default shall be **dead.letter** in the directory named by the *HOME* variable. The behavior of *mailx* in saving partial messages is unspecified if the User Portability Utilities option is not supported and *DEAD* is not defined with the value **/dev/null**.

EDITOR Determine the name of a utility to invoke when the **edit** (see Commands in mailx, on page 3112) or **~e** (see Command Escapes in mailx, on page 3122) command is used. The default editor is unspecified. XSI On XSI-conformant systems it is *ed*. The effects of this variable are unspecified if the User Portability Utilities option is not supported.

HOME Determine the pathname of the user's home directory.

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the handling of case-insensitive address and header field name comparisons.

LC_TIME This variable may determine the format and contents of the date and time strings written by *mailx*. This volume of POSIX.1-2024 specifies the effects of this variable only for systems supporting the User Portability Utilities option.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

LISTER Determine a string representing the command for writing the contents of the **folder** directory to standard output when the **folders** command is given (see **folders** in Commands in mailx, on page 3112). Any string acceptable as a *command_string* operand to the *sh* **−c** command shall be valid. If this variable is null or not set, the output command shall be *ls*. The effects of this variable are unspecified if the User Portability Utilities option is not supported.

MAILRC Determine the pathname of the user start-up file. The default shall be **.mailrc** in the directory referred to by the *HOME* environment variable. The behavior of *mailx* is unspecified if the User Portability Utilities option is not supported and *MAILRC* is not defined with the value **/dev/null**.

MBOX Determine a pathname of the file to save messages from the system mailbox that have been read. The **exit** command shall override this function, as shall saving the message explicitly in another file. The default shall be **mbox** in the directory named by the *HOME* variable. The effects of this variable are unspecified if the User Portability Utilities option is not supported.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

PAGER Determine a string representing an output filtering or pagination command for writing the output to the terminal. Any string acceptable as a *command_string* operand to the *sh* **−c** command shall be valid. When standard output is a terminal device, the message output shall be piped through the command if the *mailx* internal variable **crt** is set to a value less than the total number of lines in the message; see Internal Variables in mailx (on page 3109). When standard output is not a terminal device, it is unspecified whether the message output is written directly to standard output or is subject to pagination. If the *PAGER* variable is null or not set, the paginator shall be either *more* or another paginator utility documented in the system documentation. The effects of this variable are unspecified if the User Portability Utilities option is not supported.

SHELL Determine the name of a preferred command interpreter. The default shall be *sh*. The effects of this variable are unspecified if the User Portability Utilities option is not supported.

TERM If the internal variable **screen** is not specified, determine the name of the terminal type to indicate in an unspecified manner the number of lines in a screenful of header summaries. If *TERM* is not set or is set to null, an unspecified default terminal type shall be used and the value of a screenful is unspecified. The effects of this variable are unspecified if the User Portability Utilities option is not supported.

TZ This variable may determine the timezone used to calculate date and time strings written by *mailx*. If *TZ* is unset or null, an unspecified default timezone shall be used.

VISUAL Determine a pathname of a utility to invoke when the **visual** command (see Commands in mailx, on page 3112) or **~v** command-escape (see Command Escapes in mailx, on page 3122) is used. If this variable is null or not set, the full-screen editor shall be *vi*. The effects of this variable are unspecified if the User Portability Utilities option is not supported.

ASYNCHRONOUS EVENTS

When *mailx* is in Send Mode and standard input is not a terminal, it shall take the standard action for all signals.

UP In Receive Mode, or in Send Mode when standard input is a terminal, if a SIGINT signal is received:

UP 1. If in command mode, the current command, if there is one, shall be aborted, and a command-mode prompt shall be written.

2. If in input mode:

UP a. If **ignore** is set, *mailx* shall write `"@\n"`, discard the current input line, and continue processing, bypassing the message-abort mechanism described in item 2b.

104399 UP b. If the interrupt was received while sending mail, either when in Receive Mode or
104400 in Send Mode, a message shall be written, and another subsequent interrupt, with
104401 no other intervening characters typed, shall be required to abort the mail message.
104402 UP If in Receive Mode and another interrupt is received, a command-mode prompt
104403 shall be written. If in Send Mode and another interrupt is received, *mailx* shall
104404 terminate with a non-zero status.

104405 In both cases listed in item b, if the message is not empty:

104406 UP i. If **save** is enabled and the file named by *DEAD* can be created, the message
104407 shall be written to the file named by *DEAD*. If the file exists, the message
104408 shall be written to replace the contents of the file.

104409 UP ii. If **save** is not enabled, or the file named by *DEAD* cannot be created, the
104410 message shall not be saved.

104411 The *mailx* utility shall take the standard action for all other signals.

104412 **STDOUT**

104413 In command and input modes, all output, including prompts and messages, shall be written to
104414 standard output.

104415 **STDERR**

104416 The standard error shall be used only for diagnostic messages.

104417 **OUTPUT FILES**

104418 Various *mailx* commands and command escapes can create or add to files, including the **mbox**,
104419 the dead-letter file, and secondary mailboxes. When *mailx* is used as described in this volume of
104420 POSIX.1-2024, these files shall be text files, formatted as follows:

104421 `line beginning with` **`From<space>`**
104422 **`[`**`one or more` *`header fields`*; see Commands in mailx (on page 3112)**`]`**
104423 *`empty line`*
104424 **`[`**`zero or more` *`body lines`*
104425 *`empty line`***`]`**
104426 **`[`**`line beginning with` **`From<space>...]`**

104427 where each message begins with the **From <space>** line shown, preceded by the beginning of
104428 the file or an empty line. (The **From <space>** line is considered to be part of the message header,
104429 but not one of the header fields referred to in Commands in mailx (on page 3112); thus, it shall
104430 not be affected by the **discard**, **ignore**, or **retain** commands.) The formats of the remainder of the
104431 **From <space>** line and any additional header lines are unspecified, except that none shall be
104432 empty. The format of a message body line is also unspecified, except that no line following an
104433 empty line shall start with **From <space>**; *mailx* shall modify any such user-entered message
104434 body lines (following an empty line and beginning with **From <space>**) by adding one or more
104435 characters to precede the `'F'`; it may add these characters to **From <space>** lines that are not
104436 preceded by an empty line.

104437 When a message from the system mailbox or entered by the user is not a text file, it is
104438 implementation-defined how such a message is stored in files written by *mailx*.

104439 **EXTENDED DESCRIPTION**

104440 UP The functionality in the entire EXTENDED DESCRIPTION section shall be provided on
104441 implementations supporting the User Portability Utilities option. The functionality described in
104442 this section shall be provided on implementations that support the User Portability Utilities
104443 option (and the rest of this section is not further shaded for this option).

The *mailx* utility need not support for all character encodings in all circumstances. For example, inter-system mail may be restricted to 7-bit data by the underlying network, 8-bit data need not be portable to non-internationalized systems, and so on. Under these circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used.

When *mailx* is invoked using one of the Receive Mode synopsis forms, it shall write a page of header-summary lines (if **–N** was not specified and there are messages, see below), followed by a prompt indicating that *mailx* can accept regular commands (see Commands in mailx, on page 3112); this is termed *command mode*. The page of header-summary lines shall contain the first new message if there are new messages, or the first unread message if there are unread messages, or the first message. When *mailx* is invoked using the Send Mode synopsis and standard input is a terminal, if no subject is specified on the command line and the **asksub** variable is set, a prompt for the subject shall be written. At this point, *mailx* shall be in input mode. This input mode shall also be entered when using one of the Receive Mode synopsis forms and a reply or new message is composed using the **reply**, **Reply**, **followup**, **Followup**, or **mail** commands and standard input is a terminal. When the message is typed and the end of the message is encountered, the message shall be passed to the mail delivery software. Commands can be entered by beginning a line with the escape character (by default, <tilde> (**'~'**)) followed by a single command letter and optional arguments. See Commands in mailx (on page 3112) for a summary of these commands. It is unspecified what effect these commands will have if standard input is not a terminal when a message is entered using either the Send Mode synopsis, or the Read Mode commands **reply**, **Reply**, **followup**, **Followup**, or **mail**.

Note: For notational convenience, this section uses the default escape character, <tilde>, in all references and examples.

At any time, the behavior of *mailx* shall be governed by a set of environmental and internal variables. These are flags and valued parameters that can be set and cleared via the *mailx* **set** and **unset** commands.

Regular commands are of the form:

[*command***] [***msglist***] [***argument ...***]**

If no *command* is specified in command mode, **next** shall be assumed. In input mode, commands shall be recognized by the escape character, and lines not treated as commands shall be taken as input for the message.

In command mode, each message shall be assigned a sequential number, starting with 1.

All messages have a state that shall affect how they are displayed in the header summary and how they are retained or deleted upon termination of *mailx*. There is at any time the notion of a *current* message, which shall be marked by a **'>'** at the beginning of a line in the header summary. When *mailx* is invoked using one of the Receive Mode synopsis forms, the current message shall be the first new message, if there is a new message, or the first unread message if there is an unread message, or the first message if there are any messages, or unspecified if there are no messages in the mailbox. Each command that takes an optional list of messages (*msglist*) or an optional single message (*message*) on which to operate shall leave the current message set to the highest-numbered message of the messages specified, unless the command deletes messages, in which case the current message shall be set to the first undeleted message (that is, a message not in the deleted state) after the highest-numbered message deleted by the command, if one exists, or the first undeleted message before the highest-numbered message deleted by the command, if one exists, or to an unspecified value if there are no remaining undeleted messages. All messages shall be in one of the following states:

104491 *new* The message is present in the system mailbox and has not been viewed by the user
104492 or moved to any other state. Messages in state *new* when *mailx* quits shall be
104493 retained in the system mailbox.

104494 *unread* The message has been present in the system mailbox for more than one invocation
104495 of *mailx* and has not been viewed by the user or moved to any other state.
104496 Messages in state *unread* when *mailx* quits shall be retained in the system mailbox.

104497 *read* The message has been processed by one of the following commands: **~f**, **~m**, **~F**, **~M**,
104498 **copy**, **mbox**, **next**, **pipe**, **print**, **Print**, **top**, **type**, **Type**, **undelete**. The **dp** and **dt**
104499 commands shall also cause the message they write, if any, to be marked as *read*. If
104500 the **autoprint** variable is set, the **delete** command shall also cause the message it
104501 writes, if any, to be marked as *read*. Messages that are in the system mailbox and in
104502 state *read* when *mailx* quits shall be saved in the **mbox**, unless the internal variable
104503 **hold** was set. Messages that are in the **mbox** or in a secondary mailbox and in state
104504 *read* when *mailx* quits shall be retained in their current location.

104505 *deleted* The message has been processed by one of the following commands: **delete**, **dp**, **dt**.
104506 Messages in state *deleted* when *mailx* quits shall be deleted. Deleted messages shall
104507 be ignored until *mailx* quits or changes mailboxes or they are specified to the
104508 undelete command; for example, the message specification */string* shall only
104509 search the **Subject** header fields of messages that have not yet been deleted, unless
104510 the command operating on the list of messages is **undelete**. No deleted message
104511 or deleted message header shall be displayed by any *mailx* command other than
104512 **undelete**.

104513 *preserved* The message has been processed by a **preserve** command. When *mailx* quits, the
104514 message shall be retained in its current location.

104515 *saved* The message has been processed by one of the following commands: **save** or **write**.
104516 If the current mailbox is the system mailbox, and the internal variable **keepsave** is
104517 set, messages in the state saved shall be saved to the file designated by the *MBOX*
104518 variable (see the ENVIRONMENT VARIABLES section). If the current mailbox is
104519 the system mailbox, messages in the state *saved* shall be deleted from the current
104520 mailbox, when the **quit** or **file** command is used to exit the current mailbox.

104521 The header-summary line for each message shall indicate the state of the message.

104522 Many commands take an optional list of messages (*msglist*) on which to operate, which defaults
104523 to the current message. A *msglist* is a list of message specifications separated by <blank>
104524 characters, which can include:

104525 `n` Message number *n*.

104526 `+` The next undeleted message, or the next deleted message for the **undelete** command.

104527 `–` The next previous undeleted message, or the next previous deleted message for the
104528 **undelete** command.

104529 `.` The current message.

104530 `^` The first undeleted message, or the first deleted message for the **undelete** command.

104531 `$` The last message.

104532 `*` All messages.

104533 `n-m` An inclusive range of message numbers.

104534 *address* All messages from *address*; any address as shown in a header summary shall be
104535 matchable in this form.

104536 */string* All messages with *string* in the **Subject** header field (case ignored).

104537 `:c` All messages of type *c*, where *c* shall be one of:

104538 `d` Deleted messages.

104539 `n` New messages.

104540 `o` Old messages (any not in state *read* or *new*).

104541 `r` Read messages.

104542 `u` Unread messages.

104543 Other commands take an optional message (*message*) on which to operate, which defaults to the
104544 current message. All of the forms allowed for *msglist* are also allowed for *message*, but if more
104545 than one message is specified, only the first shall be operated on.

104546 Other arguments are usually arbitrary strings whose usage depends on the command involved.

104547 **Start-Up in mailx**

104548 At start-up time, *mailx* shall take the following steps in sequence:

104549 1. Establish all variables at their stated default values.

104550 2. Process command line options, overriding corresponding default values.

104551 3. Import any of the *DEAD*, *EDITOR*, *MBOX*, *LISTER*, *PAGER*, *SHELL*, or *VISUAL* variables
104552 that are present in the environment, overriding the corresponding default values.

104553 4. Read *mailx* commands from an unspecified system start-up file, unless the **−n** option is
104554 given, to initialize any internal *mailx* variables and aliases.

104555 5. Process the user start-up file of *mailx* commands named in the user *MAILRC* variable.

104556 Most regular *mailx* commands are valid inside start-up files, the most common use being to set
104557 up initial display options and alias lists. The following commands shall be invalid in a start-up
104558 file: **!**, **edit**, **hold**, **mail**, **preserve**, **reply**, **Reply**, **Save**, **shell**, **visual**, **Copy**, **followup**, and
104559 **Followup**. Any errors in a start-up file shall either cause *mailx* to terminate with a diagnostic
104560 message and a non-zero status or to continue after writing a diagnostic message, ignoring the
104561 remainder of the lines in the file.

104562 A blank line in a start-up file shall be ignored.

104563 **Internal Variables in mailx**

104564 The following variables are internal *mailx* variables. Each internal variable can be set via the
104565 *mailx* **set** command at any time. The **unset** and **set no***name* commands can be used to erase
104566 variables.

104567 In the following list, variables shown as:

104568 `variable`

104569 represent Boolean values. Variables shown as:

104570 `variable=`*`value`*

shall be assigned string or numeric values. For string values, the rules in Commands in mailx (on page 3112) concerning filenames and quoting shall also apply.

The defaults specified here may be changed by the unspecified system start-up file unless the user specifies the –**n** option.

allnet All network names whose login name components match shall be treated as identical. This shall cause the *msglist* message specifications to behave similarly. The default shall be **noallnet**. See also the **alternates** command and the **metoo** variable.

append Append messages to the end of the **mbox** file upon termination instead of placing them at the beginning. The default shall be **noappend**. This variable shall not affect the **save** command when saving to **mbox**.

ask, **asksub** Prompt for a value for the **Subject** header field on outgoing mail if one is not specified on the command line with the –**s** option. The **ask** and **asksub** forms are synonyms; the system shall refer to **asksub** and **noasksub** in its messages, but shall accept **ask** and **noask** as user input to mean **asksub** and **noasksub**. It shall not be possible to set both **ask** and **noasksub**, or **noask** and **asksub**. The default shall be **asksub**, but no prompting shall be done if standard input is not a terminal.

askbcc Prompt for the blind copy list. The default shall be **noaskbcc**.

askcc Prompt for the copy list. The default shall be **noaskcc**.

autoprint Enable automatic writing of messages after **delete** and **undelete** commands. The default shall be **noautoprint**.

bang Enable the special-case treatment of <exclamation-mark> characters (`'!'`) in **!** commands and **~!***command* escapes; see the **Invoke Shell Command** command and Command Escapes in mailx (on page 3122). The default shall be **nobang**, disabling the expansion of `'!'` in the *command* argument to the **!** command and the **~!***command* escape.

cmd=*command*
Set the default command to be invoked by the **pipe** command. The default shall be **nocmd**.

crt=*number* Paginate message output as described for the *PAGER* variable. The default shall be **nocrt**, disabling this pagination. If it is set to null, the value used is implementation-defined.

XSI **debug** Enable verbose diagnostics for debugging. Messages are not delivered. The default shall be **nodebug**.

dot When **dot** is set, a <period> on a line by itself during message input from a terminal shall also signify end-of-file (in addition to normal end-of-file). The default shall be **nodot**. If **ignoreeof** is set (see below), a setting of **nodot** shall be ignored and <period> and the **~.** command escape are the only methods to terminate input mode.

escape=*c* Set the command escape character to be the character `'c'`. By default, the command escape character shall be <tilde>. If **escape** is unset, <tilde> shall be used; if it is set to null, command escaping shall be disabled.

flipr Reverse the meanings of the **R** and **r** commands. The default shall be **noflipr**.

folder=*directory*
The default directory for saving mail files. User-specified filenames beginning with a <plus-sign> (`'+'`) shall be expanded by preceding the filename with this directory name to obtain the real pathname. If *directory* does not start with a <slash> (`'/'`), the contents of *HOME* shall be prefixed to it. The default shall be **nofolder**. If **folder** is unset, user-specified filenames beginning with `'+'` shall refer to files in the current directory that begin with the literal `'+'` character. See also **outfolder** below. The **folder** value need not affect the processing of the files named in *MBOX* and *DEAD*.

header Enable writing of the header summary when entering *mailx* in Receive Mode. The default shall be **header**.

hold Disable message moving of read messages from the system mailbox to the **mbox** save file upon normal program termination or folder change. This automatic mail management is complemented with the commands **hold** (and **preserve**), **mbox**, and **touch**, which partially override the **hold** variable. The default shall be **nohold**.

ignore Ignore interrupts while entering messages. The default shall be **noignore**.

ignoreeof Ignore normal end-of-file during message input. Input can be terminated only by entering a <period> (`'.'`) on a line by itself or by the **~.** command escape. The default shall be **noignoreeof**. See also **dot** above.

indentprefix=*string*
A string that shall be added as a prefix to each line that is inserted into the message by the **~m** command escape. This variable shall default to one <tab>.

keep When a system mailbox, secondary mailbox, or **mbox** is empty, truncate it to zero length instead of removing it. The default shall be **nokeep**.

keepsave Keep the messages that have been saved from the system mailbox into other files in the file designated by the variable *MBOX*, instead of deleting them. The default shall be **nokeepsave**.

metoo Suppress the deletion of the user's login name from the recipient list when replying to a message or sending to a group. The default shall be **nometoo**.

XSI **onehop** When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (that is, one hop away). The default shall be **noonehop**.

outfolder Cause the files used to record outgoing messages to be located in the directory specified by the **folder** variable unless the pathname is absolute. The default shall be **nooutfolder**. See the **record** variable.

page Insert a <form-feed> after each message sent through the pipe created by the **pipe** command. The default shall be **nopage**.

prompt=*string*
Set the command-mode prompt to *string*. If *string* is null or if **noprompt** is set, no prompting shall occur. The default shall be to prompt with the string `"? "`.

quiet Refrain from writing the opening message and version when entering *mailx*. The default shall be **noquiet**.

104658 **record**=*file* Record all outgoing mail in the file with the pathname *file*. The default shall be
104659 **norecord**. See also **outfolder** above.

104660 **save** Enable saving of messages in the dead-letter file on interrupt or delivery error. See
104661 the variable *DEAD* for the location of the dead-letter file. The default shall be **save**.

104662 **screen**=*number*
104663 Set the number of lines in a screenful of headers for the **headers** and **z** commands.
104664 If **screen** is not specified, a value based on the terminal type identified by the
104665 *TERM* environment variable, the window size, the baud rate, or some combination
104666 of these shall be used. The default shall be **noscreen**.

104667 **sendwait** Wait for the background mailer to finish before returning. The default shall be
104668 **nosendwait**.

104669 **showto** When the sender of the message was the user who is invoking *mailx*, write the
104670 information from the **To** field instead of the **From** field in the header summary. The
104671 default shall be **noshowto**.

104672 **sign**=*string* Set the variable inserted into the text of a message when the **~a** command escape is
104673 given. The default shall be **nosign**. The character sequences `'\t'` and `'\n'` shall
104674 be recognized in the variable as <tab> and <newline> characters, respectively. (See
104675 also **~i** in Command Escapes in mailx (on page 3122).)

104676 **Sign**=*string* Set the variable inserted into the text of a message when the **~A** command escape is
104677 given. The default shall be **noSign**. The character sequences `'\t'` and `'\n'` shall
104678 be recognized in the variable as <tab> and <newline> characters, respectively.

104679 **toplines**=*number*
104680 Set the number of lines of the message to write with the **top** command. The default
104681 shall be 5.

104682 **Commands in mailx**

104683 The following *mailx* commands shall be provided. In the following list, *header* refers to lines from
104684 the message header, as shown in the OUTPUT FILES section. *Header field* refers to a line or set of
104685 lines within the header that begins with one or more non-white-space characters immediately
104686 followed by a <colon> and white space, and continuing up to and including a <newline> that
104687 immediately precedes either the next line beginning with a non-white-space character or an
104688 empty line. *Field name* refers to the portion of a header field prior to the first <colon>.

104689 For each of the commands listed below, the command can be entered as the abbreviation (those
104690 characters in the Synopsis command word preceding the `'['`), the full command (all characters
104691 shown for the command word, omitting the `'['` and `']'`), or any truncation of the full
104692 command down to the abbreviation. For example, the **exit** command (shown as **ex[it]** in the
104693 Synopsis) can be entered as **ex**, **exi**, or **exit**.

104694 The arguments to commands can be quoted, using the following methods:

104695 • An argument can be enclosed between paired double-quotes (`""`) or single-quotes (`''`);
104696 any white space, shell word expansion, or <backslash> characters within the quotes shall
104697 be treated literally as part of the argument. A double-quote shall be treated literally within
104698 single-quotes and *vice versa*. These special properties of the <quotation-mark> characters
104699 shall occur only when they are paired at the beginning and end of the argument.

104700 • A <backslash> outside of the enclosing quotes shall be discarded and the following
104701 character treated literally as part of the argument.

104702 • An unquoted <backslash> at the end of a command line shall be discarded and the next
104703 line shall continue the command.

104704 Filenames, where expected, shall be subjected to the following transformations, in sequence:

104705 • If the filename begins with an unquoted <plus-sign>, and the **folder** variable is defined
104706 (see the **folder** variable), the <plus-sign> shall be replaced by the value of the **folder**
104707 variable followed by a <slash>. If the **folder** variable is unset or is set to null, the filename
104708 shall be unchanged.

104709 • Shell word expansions shall be applied to the filename (see Section 2.6, on page 2483). If
104710 more than a single pathname results from this expansion and the command is expecting
104711 one file, the effects are unspecified.

104712 **Declare Aliases**

104713 *Synopsis*: `a[lias] [`*`alias`*` [`*`address...`*`]]`
104714 `g[roup] [`*`alias`*` [`*`address...`*`]]`

104715 Add the given addresses to the alias specified by *alias*. The names shall be substituted when
104716 *alias* is used as a recipient address specified by the user in an outgoing message (that is, other
104717 recipients addressed indirectly through the **reply** command shall not be substituted in this
104718 manner). Mail address alias substitution shall apply only when the alias string is used as a full
104719 address; for example, when **hlj** is an alias, *hlj@posix.com* does not trigger the alias substitution.
104720 Recursive expansion of an alias group member can be prevented by prefixing it with an
104721 unquoted <backslash>. If no arguments are given, write a listing of the current aliases to
104722 standard output. If only an *alias* argument is given, write a listing of the specified alias to
104723 standard output. These listings need not reflect the same order of addresses that were entered.

104724 **Declare Alternatives**

104725 *Synopsis*: `alt[ernates]` *`name...`*

104726 Declare a list of alternative addresses for the address consisting of the user's login name. When
104727 responding to a message, these alternative addresses shall be removed from the list of recipients.
104728 The comparison of addresses shall be performed in a case-insensitive manner. With no
104729 arguments, **alternates** shall write the current list of alternative addresses.

104730 **Change Current Directory**

104731 *Synopsis*: `cd [`*`directory`*`]`
104732 `ch[dir] [`*`directory`*`]`

104733 Change directory. If *directory* is not specified, the contents of *HOME* shall be used.

104734 **Copy Messages**

104735 *Synopsis*: `c[opy] [`*`file`*`]`
104736 `c[opy] [`*`msglist`*`]` *`file`*
104737 `C[opy] [`*`msglist`*`]`

104738 Copy messages to the file named by the pathname *file* without marking the messages as saved.
104739 Otherwise, it shall be equivalent to the **save** command.

104740 In the capitalized form, save the specified messages in a file whose name is derived from the
104741 author of the message to be saved, without marking the messages as saved. Otherwise, it shall
104742 be equivalent to the **Save** command.

104743 **Delete Messages**

104744 *Synopsis*: `d[elete] [`*`msglist`*`]`

104745 Mark messages for deletion from the mailbox. The deletions shall not occur until *mailx* quits (see
104746 the **quit** command) or changes mailboxes (see the **folder** command). If **autoprint** is set and there
104747 are messages remaining after the **delete** command, the current message shall be written as
104748 described for the **print** command (see the **print** command); otherwise, the *mailx* prompt shall be
104749 written.

104750 **Discard Header Fields**

104751 *Synopsis*: `di[scard] [`*`field-name...`*`]`
104752 `ig[nore] [`*`field-name...`*`]`

104753 Suppress header fields with the specified field names when writing messages. Specified *field-*
104754 *name* arguments shall be added to the list of suppressed field names. Examples of field names to
104755 ignore are **status** and **cc**. The header fields shall be included when the message is saved. The
104756 **Print** and **Type** commands shall override this command. The comparison of field names shall be
104757 performed in a case-insensitive manner. If no arguments are specified, write a list of the
104758 currently suppressed field names to standard output; the listing need not reflect the same order
104759 of field names that were entered.

104760 If both **retain** and **discard** commands are given, **discard** commands shall be ignored.

104761 **Delete Messages and Display**

104762 *Synopsis*: `dp [`*`msglist`*`]`
104763 `dt [`*`msglist`*`]`

104764 Delete the specified messages as described for the **delete** command, except that the **autoprint**
104765 variable shall have no effect, and the current message shall be written only if it was set to a
104766 message after the last message deleted by the command. Otherwise, an informational message
104767 to the effect that there are no further messages in the mailbox shall be written, followed by the
104768 *mailx* prompt.

104769 **Echo a String**

104770 *Synopsis*: `ec[ho]` *`string ...`*

104771 Echo the given strings, equivalent to the shell *echo* utility.

104772 **Edit Messages**

104773 *Synopsis*: `e[dit] [`*`msglist`*`]`

104774 Edit the given messages. The messages shall be placed in a temporary file and the utility named
104775 by the *EDITOR* variable is invoked to edit each file in sequence. The default *EDITOR* is
104776 unspecified.

104777 The **edit** command does not modify the contents of those messages in the mailbox.

104778 **Exit**

104779 *Synopsis*: `ex[it]`
104780 `x[it]`

104781 Exit from *mailx* without performing automatic message moving, or any other management tasks.
104782 See also **quit**.

104783 **Change Folder**

104784 *Synopsis*: `fi[le] [`*`file`*`]`
104785 `fold[er] [`*`file`*`]`

104786 If no argument is given, write the name and status of the current mailbox. Otherwise, close the
104787 current file of messages after performing actions as specified for the **quit** command (except for
104788 terminating *mailx*) and then read in the file named by the pathname *file*. The behavior is
104789 unspecified if *file* is not a valid **mbox**.

104790 Several unquoted special characters shall be recognized when used as *file* names, with the
104791 following substitutions:

104792 `%` The system mailbox for the invoking user.

104793 `%`*`user`* The system mailbox for *user*.

104794 `#` The previous file.

104795 `&` The current **mbox**.

104796 `+`*`file`* The named file in the **folder** directory. (See the **folder** variable.)

104797 The default file shall be the current mailbox.

104798 **Display List of Folders**

104799 *Synopsis*: `folders`

104800 Write the names of the files in the directory set by the **folder** variable. The command specified
104801 by the *LISTER* environment variable shall be used (see the ENVIRONMENT VARIABLES
104802 section).

104803 **Follow Up Specified Messages**

104804 *Synopsis*: `fo[llowup] [`*`message`*`]`
104805 `F[ollowup] [`*`msglist`*`]`

104806 The **followup** and **Followup** commands shall be equivalent to **reply** and **Reply**, respectively,
104807 except that:

104808 • They shall ignore the **record** variable.

104809 • The **followup** command shall record the response in a file whose name is derived from the
104810 author of the *message*.

104811 • The **Followup** command shall record the response in a file whose name is derived from the
104812 author of the first message in the *msglist*.

104813 See also the **save** and **copy** commands and **outfolder**.

104814 **Display Header Summary for Specified Messages**

104815 *Synopsis*: `f[rom] [`*msglist*`]`

104816 Write the header summary for the specified messages.

104817 **Display Header Summaries**

104818 *Synopsis*: `h[eaders] [`*message*`]`

104819 Write the page of header summaries that includes the message specified. If the *message*
104820 argument is not specified, the current message shall not change. However, if the *message*
104821 argument is specified, the current message shall become the message that appears at the top of
104822 the page of header summaries that includes the message specified. The **screen** variable sets the
104823 number of header summaries per page. See also the **z** command.

104824 **Help**

104825 *Synopsis*: `hel[p]`
104826 `?`

104827 Write a summary of commands.

104828 **Hold Messages**

104829 *Synopsis*: `ho[ld] [`*msglist*`]`
104830 `pre[serve] [`*msglist*`]`

104831 Allowed only in the system mailbox. Mark the messages in *msglist* to be preserved, as if the **hold**
104832 variable were set, upon normal termination or when the folder is changed. This shall override
104833 any commands that might previously have marked the messages to be deleted, and only the
104834 **delete**, **dp**, or **dt**, as well as the **mbox** and **touch** commands, shall remove the *preserve* mark of a
104835 message.

104836 **Execute Commands Conditionally**

104837 *Synopsis*: `i[f] s|r`
104838 *mail-commands*
104839 `el[se]`
104840 *mail-commands*
104841 `en[dif]`

104842 Execute commands conditionally, where **if s** executes the following *mail-commands*, up to an
104843 **else** or **endif**, if the program is in Send Mode, and **if r** shall cause the *mail-commands* to be
104844 executed only in Receive Mode.

104845 **List Available Commands**

104846 *Synopsis*: `l[ist]`

104847 Write a list of all commands available. No explanation shall be given.

104848 **Mail a Message**

104849 *Synopsis*: `m[ail] address...`

104850 Mail a message to the specified addresses or aliases.

104851 **Direct Messages to mbox**

104852 *Synopsis*: `mb[ox] [msglist]`

104853 Allowed only in the system mailbox. Arrange for the given messages to end up in the secondary
104854 mailbox, overriding a possibly set **hold** variable, upon normal termination or when the folder is
104855 changed. Overrides a former **hold** or **preserve** request. See *MBOX* in the ENVIRONMENT
104856 VARIABLES section. See also the **exit** and **quit** commands.

104857 **Process Next Specified Message**

104858 *Synopsis*: `n[ext] [message]`

104859 If the current message has not been written (for example, by the **print** command) since *mailx*
104860 started or since any other message was the current message, behave as if the **print** command
104861 was entered. Otherwise, if there is an undeleted message after the current message, make it the
104862 current message and behave as if the **print** command was entered. Otherwise, an informational
104863 message to the effect that there are no further messages in the mailbox shall be written, followed
104864 by the *mailx* prompt. Should the current message location be the result of an immediately
104865 preceding **hold**, **mbox**, **preserve**, or **touch** command, **next** shall act as if the current message has
104866 already been written.

104867 **Pipe Message**

104868 *Synopsis*: `pi[pe] [[msglist] command]`
104869 `| [[msglist] command]`

104870 Pipe the messages through the given *command* by invoking the command interpreter specified
104871 by *SHELL* with three arguments: `"-c"`, `"--"`, and *command*. (See also *sh* **-c**.) The application
104872 shall ensure that the command is given as a single argument. Quoting, described previously, can
104873 be used to accomplish this. If no arguments are given, the current message shall be piped
104874 through the command specified by the value of the **cmd** variable. If the **page** variable is set, a
104875 <form-feed> shall be inserted after each message.

104876 **Display Message with Header**

104877 *Synopsis*: `P[rint] [msglist]`
104878 `T[ype] [msglist]`

104879 Write the specified messages, including all header fields, to standard output. This command
104880 shall override suppression of header fields by the **discard**, **ignore**, and **retain** commands. If **crt** is
104881 set, the output shall be paginated as described for the *PAGER* variable.

104882 **Display Message**

104883 *Synopsis*: `p[rint] [`*msglist*`]`
104884 `t[ype] [`*msglist*`]`

104885 Write the specified messages to standard output. If **crt** is set, the output shall be paginated as
104886 described for the *PAGER* variable.

104887 **Quit**

104888 *Synopsis*: `q[uit]`
104889 *end-of-file*

104890 Terminate *mailx* normally, performing automatic message moving as specified in the description
104891 of the variable **hold**, deleting messages that have been explicitly saved (unless **keepsave** is set),
104892 discarding messages that have been deleted, and saving all remaining messages in the mailbox.

104893 **Reply to a Message or a Message List**

104894 *Synopsis*: `r[eply] [`*message*`]`
104895 `r[espond] [`*message*`]`
104896 `R[eply] [`*msglist*`]`
104897 `R[espond] [`*msglist*`]`

104898 Mail a reply message to one or more addresses taken from certain header fields in the specified
104899 message or message list. If the **flipr** variable is unset, these commands shall behave as described
104900 below. If the **flipr** variable is set, commands in the lowercase form shall behave as described
104901 below for commands in the capitalized form, and *vice versa*; the synopsis forms shown above
104902 shall also be swapped accordingly.

104903 The recipients of the reply message shall be determined by first constructing an initial list of
104904 recipients and then modifying it to form the list that is in effect when *mailx* enters input mode.

104905 In the capitalized form, the initial list of recipients shall be taken from the header of each
104906 message in the *msglist* as follows:

104907 • If the header contains a **Reply-To** field, the address or addresses in that field shall be
104908 added to the list.

104909 • Otherwise, the address or addresses in the **From** field of the header shall be added to the
104910 list.

104911 In the lowercase form, the initial list of recipients shall be taken from the header of the *message* as
104912 follows:

104913 • If the header does not contain a **Reply-To** field, all of the addresses in the **From**, **To**, and **Cc**
104914 fields shall be included in the list.

104915 • Otherwise, it is implementation-defined whether all of the addresses in the **Reply-To**, **To**,
104916 and **Cc** fields are included in the list or only the address or addresses in the **Reply-To** field.

104917 The initial list of recipients shall be marked for placement in the header fields of the reply
104918 message as follows. Recipient addresses taken from a **From** or **Reply-To** header field shall be
104919 marked for placement in the **To** field of the reply message. Recipient addresses taken from a **Cc**
104920 header field shall be marked for placement in the **Cc** field of the reply message. Recipient
104921 addresses taken from a **To** header field shall be marked for placement in either the **To** or the **Cc**
104922 field of the reply message. Implementations shall provide a way to place them in the **To** field.
104923 Implementations may, but need not, provide an implementation-defined way to place them in
104924 the **Cc** field.

The modifications applied to the initial list of recipients shall be as follows:

- If the **metoo** variable is unset, addresses consisting of the login name of the user and any alternative addresses declared using the **alternates** command shall be removed from the list.
- The set of recipients marked for placement in the **To** header field of the reply message shall have duplicates within that set removed.
- The set of recipients marked for placement in the **Cc** header field of the reply message shall have duplicates within that set removed and may have recipients that are also marked for placement in the **To** field removed.

The values for the **To** and **Cc** header fields of the reply message shall be constructed from the addresses in the modified list of recipients that are marked for placement in each of those fields.

The value for the **Subject** header field of the reply message shall be formed from the value of the **Subject** header field of the *message* or the first message in *msglist* by prefixing it with **Re:**<space>, unless it already begins with that string.

The values of the **To**, **Cc**, and **Subject** header fields set as described above can be modified by the user after *mailx* enters input mode through the use of the **~t**, **~c**, **~s**, and **~h** command escapes.

If **record** is set to a pathname, the response shall be saved at the end of that file.

Retain Header Fields

Synopsis: `ret[ain] [`*`field-name...`*`]`

Retain header fields with the specified field names when writing messages. This command shall override all **discard** and **ignore** commands. The comparison of field names shall be performed in a case-insensitive manner. If no arguments are specified, write a list of the currently retained field names to standard output; the listing need not reflect the same order of field names that were entered.

Save Messages

Synopsis:
```
s[ave] [file]
s[ave] [msglist] file
S[ave] [msglist]
```

Save the specified messages in the file named by the pathname *file*, or the **mbox** if the *file* argument is omitted. The file shall be created if it does not exist; otherwise, the messages shall be appended to the file. The message shall be put in the state *saved*, and shall behave as specified in the description of the *saved* state when the current mailbox is exited by the **quit** or **file** command.

In the capitalized form, save the specified messages in a file whose name is derived from the author of the first message. The name of the file shall be taken to be the author's name with all network addressing stripped off. See also the **Copy**, **followup**, and **Followup** commands and **outfolder** variable.

104962 **Set Variables**

104963 *Synopsis*: `se[t] [`*name*`[=[`*string*`]] ...] [`*name*`=`*number* `...] [no`*name* `...]`

104964 Define one or more variables called *name*. The variable can be given a null, string, or numeric
104965 value. Quoting and <backslash>-escapes can occur anywhere in *string*, as described previously,
104966 as if the *string* portion of the argument were the entire argument. The forms *name* and *name*=
104967 shall be equivalent to *name*="" for variables that take string values. The **set** command without
104968 arguments shall write a list of all defined variables and their values. The **no** *name* form shall be
104969 equivalent to **unset** *name*.

104970 **Invoke a Shell**

104971 *Synopsis*: `sh[ell]`

104972 Invoke an interactive command interpreter (see also *SHELL*).

104973 **Display Message Size**

104974 *Synopsis*: `si[ze] [`*msglist*`]`

104975 Write the size in bytes of each of the specified messages.

104976 **Read mailx Commands From a File**

104977 *Synopsis*: `so[urce]` *file*

104978 Read and execute commands from the file named by the pathname *file* and return to command
104979 mode.

104980 **Display Beginning of Messages**

104981 *Synopsis*: `to[p] [`*msglist*`]`

104982 Write the top few lines of each of the specified messages. If the **toplines** variable is set, it is taken
104983 as the number of lines to write. The default shall be 5.

104984 **Touch Messages**

104985 *Synopsis*: `tou[ch] [`*msglist*`]`

104986 Allowed only in the system mailbox. Touch the specified messages. Unless overridden by the
104987 **hold** variable, any message in *msglist* that is not specifically deleted nor saved in a file shall be
104988 placed in the **mbox** upon normal termination or when the folder is changed. Overrides a former
104989 **hold** or **preserve** request.

104990 **Delete Aliases**

104991 *Synopsis*: `una[lias] [`*alias*`]...`

104992 Delete the specified alias names. If a specified alias does not exist, the results are unspecified.

104993 **Undelete Messages**

104994 *Synopsis*: `u[ndelete] [`*`msglist`*`]`

104995 Change the state of the specified messages from deleted to read. If **autoprint** is set, the last
104996 message of those restored shall be written. If *msglist* is not specified, the message shall be
104997 selected as follows:

104998 • If there are any deleted messages that follow the current message, the first of these shall be
104999 chosen.

105000 • Otherwise, the last deleted message that also precedes the current message shall be chosen.

105001 **Unset Variables**

105002 *Synopsis*: `uns[et]` *`name...`*

105003 Cause the specified variables to be erased.

105004 **Edit Message with Full-Screen Editor**

105005 *Synopsis*: `v[isual] [`*`msglist`*`]`

105006 Edit the given messages with a screen editor. Each message shall be placed in a temporary file,
105007 and the utility named by the *VISUAL* variable shall be invoked to edit each file in sequence. The
105008 default editor shall be *vi*.

105009 The **visual** command does not modify the contents of those messages in the mailbox.

105010 **Write Messages to a File**

105011 *Synopsis*: `w[rite] [`*`msglist`*`]` *`file`*

105012 Write the given messages to the file specified by the pathname *file*, minus the message header.
105013 Otherwise, it shall be equivalent to the **save** command.

105014 **Scroll Header Display**

105015 *Synopsis*: `z[+|-]`

105016 Scroll the header display forward (if `'+'` is specified or if no option is specified) or backward (if
105017 `'-'` is specified) one screenful. The number of header summaries written shall be set by the
105018 **screen** variable.

105019 **Invoke Shell Command**

105020 *Synopsis*: `!`*`command`*

105021 Invoke the command interpreter specified by *SHELL* with three arguments: `"-c"`, `"--"`, and
105022 *command*. (See also *sh* **−c**.) If the **bang** variable is set, each unescaped occurrence of `'!'` in
105023 *command* shall be replaced with the command executed by the previous **!** command or **~!**
105024 command escape.

105025 **Null Command**

105026 *Synopsis*: `# comment`

105027 This null command (comment) shall be ignored by *mailx*.

105028 **Display Current Message Number**

105029 *Synopsis*: `=`

105030 Write the current message number.

105031 **Command Escapes in mailx**

105032 The following commands can be entered only from input mode, by beginning a line with the
105033 escape character (by default, **<**tilde**>** (`'~'`)). See the **escape** variable description for changing
105034 this special character. The format for the commands shall be:

105035 `<escape-character><command-char><separator>`**[**`<arguments>`**]**

105036 where the **<***separator***>** can be zero or more **<**blank**>** characters.

105037 In the following descriptions, the application shall ensure that the argument *command* (but not
105038 *mailx-command)* is a shell command string. Any string acceptable to the command interpreter
105039 specified by the *SHELL* variable when it is invoked as *SHELL* **–c** *command_string* shall be valid.
105040 The command can be presented as multiple arguments (that is, quoting is not required).

105041 Command escapes that are listed with *msglist* or *mailx-command* arguments are invalid in Send
105042 Mode and produce unspecified results.

105043 **~!** *command* Invoke the command interpreter specified by *SHELL* with three arguments: `"-c"`,
105044 `"--"`, and *command*; and then return to input mode. If the **bang** variable is set,
105045 each unescaped occurrence of `'!'` in *command* shall be replaced with the command
105046 executed by the previous **!** command or **~!** command escape.

105047 **~.** Simulate end-of-file (terminate message input).

105048 **~:** *mailx-command*, **~_** *mailx-command*
105049 Perform the command-level request.

105050 **~?** Write a summary of command escapes.

105051 **~A** This shall be equivalent to **~i Sign**.

105052 **~a** This shall be equivalent to **~i sign**.

105053 **~b** *name*... Add the *name*s to the blind carbon copy (**Bcc**) list.

105054 **~c** *name*... Add the *name*s to the carbon copy (**Cc**) list.

105055 **~d** Read in the dead-letter file. See *DEAD* for a description of this file.

105056 **~e** Invoke the editor, as specified by the *EDITOR* environment variable, on the partial
105057 message.

105058 **~f [***msglist***]** Forward the specified messages. The specified messages, including their headers,
105059 shall be inserted into the current message without alteration. The header fields
105060 included in each header shall be affected by the **discard**, **ignore**, and **retain**
105061 commands.

105062 **~F** [*msglist*] This shall be the equivalent of the **~f** command escape, except that all header fields
105063 shall be included in the message headers, regardless of previous **discard**, **ignore**,
105064 and **retain** commands.

105065 **~h** If standard input is a terminal, prompt for values for the **Subject**, **To**, **Cc**, and **Bcc**
105066 header fields. Other implementation-defined header fields may also be presented
105067 for editing. If the header field is written with an initial value, it can be edited as if it
105068 had just been typed.

105069 **~i** *string* Insert the value of the named variable, followed by a <newline>, into the text of
105070 the message. If the string is unset or null, the message shall not be changed.

105071 **~m** [*msglist*] Insert the specified messages, including their headers, into the current message,
105072 prefixing non-empty lines with the string in the **indentprefix** variable. The header
105073 fields included in each header shall be affected by the **discard**, **ignore**, and **retain**
105074 commands.

105075 **~M** [*msglist*] This shall be the equivalent of the **~m** command escape, except that all header
105076 fields shall be included in the message headers, regardless of previous **discard**,
105077 **ignore**, and **retain** commands.

105078 **~p** Write the message being entered. If the message is longer than **crt** lines (see
105079 Internal Variables in mailx, on page 3109), the output shall be paginated as
105080 described for the *PAGER* variable.

105081 **~q** Quit (see the **quit** command) from input mode by simulating an interrupt. If the
105082 body of the message is not empty, the partial message shall be saved in the dead-
105083 letter file. See *DEAD* for a description of this file.

105084 **~r** *file*, **~<** *file*, **~r !***command*, **~< !***command*
105085 Read in the file specified by the pathname *file*. If the argument begins with an
105086 <exclamation-mark> (`'!'`), the rest of the string shall be taken as an arbitrary
105087 system command; the command interpreter specified by *SHELL* shall be invoked
105088 with three arguments: `"-c"`, `"--"`, and *command*. The standard output of
105089 *command* shall be inserted into the message.

105090 **~s** *string* Set the value for the **Subject** header field to *string*.

105091 **~t** *name*... Add the given *name*s to the **To** list.

105092 **~v** Invoke the full-screen editor, as specified by the *VISUAL* environment variable, on
105093 the partial message.

105094 **~w** *file* Write the partial message, without the header, onto the file named by the
105095 pathname *file*. The file shall be created or the message shall be appended to it if
105096 the file exists.

105097 **~x** Exit as with **~q**, except the message shall not be saved in the dead-letter file.

105098 **~|** *command* Pipe the body of the message through the given *command* by invoking the
105099 command interpreter specified by *SHELL* with three arguments: `"-c"`, `"--"`, and
105100 *command*. If the *command* returns a successful exit status, the standard output of
105101 the command shall replace the message. Otherwise, the message shall remain
105102 unchanged. If the *command* fails, an error message giving the exit status shall be
105103 written.

EXIT STATUS

When the **−e** option is specified, the following exit values are returned:

0 Mail was found.

>0 Mail was not found or an error occurred.

Otherwise, the following exit values are returned:

0 Successful completion; note that this status implies that any messages that *mailx* was instructed to send were all successfully either *sent* or discarded (see **−E**), but it gives no assurances that any of them were actually *delivered*.

>0 An error occurred.

CONSEQUENCES OF ERRORS

When in input mode (Receive Mode) or Send Mode:

- If an error is encountered processing an input line beginning with a <tilde> (`'~'`) character, (see Command Escapes in mailx, on page 3122), a diagnostic message shall be written to standard error, and the message being composed may be modified, but this condition shall not prevent the message from being sent.
- Other errors shall prevent the sending of the message.

When in command mode:

- Default.

APPLICATION USAGE

Delivery of messages to remote systems requires the existence of communication paths to such systems. These need not exist.

Input lines are limited to {LINE_MAX} bytes, but mailers between systems may impose more severe line-length restrictions. This volume of POSIX.1-2024 does not place any restrictions on the length of messages handled by *mailx*, and for delivery of local messages the only limitations should be the normal problems of available disk space for the target mail file. When sending messages to external machines, applications are advised to limit messages to less than 100 000 bytes because some mail gateways impose message-length restrictions.

The format of the system mailbox is intentionally unspecified. Not all systems implement system mailboxes as flat files, particularly with the advent of multimedia mail messages. Some system mailboxes may be multiple files, others records in a database. The internal format of the messages themselves is specified with the historical format from Version 7, but only after the messages have been saved in some file other than the system mailbox. This was done so that many historical applications expecting text-file mailboxes are not broken.

Some new formats for messages can be expected in the future, probably including binary data, bit maps, and various multimedia objects. As described here, *mailx* is not prohibited from handling such messages, but it must store them as text files in secondary mailboxes (unless some extension, such as a variable or command line option, is used to change the stored format). Its method of doing so is implementation-defined and might include translating the data into text file-compatible or readable form or omitting certain portions of the message from the stored output.

The **discard** and **ignore** commands are not inverses of the **retain** command. The **retain** command discards all header fields except those explicitly retained. The **discard** command

keeps all header fields except those explicitly discarded. If field names exist on the retained field names list, **discard** and **ignore** commands are ignored.

EXAMPLES

None.

RATIONALE

The standard developers felt strongly that a method for applications to send messages to specific users was necessary. The obvious example is a batch utility, running non-interactively, that wishes to communicate errors or results to a user. However, the actual format, delivery mechanism, and method of reading the message are clearly beyond the scope of this volume of POSIX.1-2024.

The intent of this command is to provide a simple, portable interface for sending messages non-interactively. It merely defines a ``front-end'' to the historical mail system. It is suggested that implementations explicitly denote the sender and recipient in the body of the delivered message. Further specification of formats for either the message envelope or the message itself were deliberately not made, as the industry is in the midst of changing from the current standards to a more internationalized standard and it is probably incorrect, at this time, to require either one.

Implementations are encouraged to conform to the various delivery mechanisms described in the CCITT X.400 standards or to the equivalent Internet standards, described in Internet Request for Comment (RFC) documents RFC 819, RFC 920, RFC 921, RFC 1123, and RFC 5322.

Many historical systems modified each body line that started with **From** by prefixing the `'F'` with `'>'`. It is unnecessary, but allowed, to do that when the string does not follow a blank line because it cannot be confused with the next header.

The **edit** and **visual** commands merely edit the specified messages in a temporary file. They do not modify the contents of those messages in the mailbox; such a capability could be added as an extension, such as by using different command names.

The restriction on the value for a **Subject** header field being {LINE_MAX}–10 bytes is based on the historical format that consumes 10 bytes for **Subject:** and the trailing **<**newline**>**. Many historical mailers that a message may encounter on other systems are not able to handle lines that long, however.

Like the utilities *logger* and *lp*, *mailx* admittedly is difficult to test. This was not deemed sufficient justification to exclude this utility from this volume of POSIX.1-2024. It is also arguable that it is, in fact, testable, but that the tests themselves are not portable.

When *mailx* is being used by an application that wishes to receive the results as if none of the User Portability Utilities option features were supported, the *DEAD* environment variable must be set to **/dev/null**. Otherwise, it may be subject to the file creations described in *mailx* ASYNCHRONOUS EVENTS. Similarly, if the *MAILRC* environment variable is not set to **/dev/null**, historical versions of *mailx* and *Mail* read initialization commands from a file before processing begins. Since the initialization that a user specifies could alter the contents of messages an application is trying to send, such applications must set *MAILRC* to **/dev/null**.

The description of *LC_TIME* uses ``may affect'' because many historical implementations do not or cannot manipulate the date and time strings in the incoming mail headers. Some header fields found in incoming mail do not have enough information to determine the timezone in which the mail originated, and, therefore, *mailx* cannot convert the date and time strings into the internal form that then is parsed by routines like *strftime*( ) that can take *LC_TIME* settings into account. Changing all these times to a user-specified format is allowed, but not required.

The paginator selected when *PAGER* is null or unset is partially unspecified to allow the System

V historical practice of using *pg* as the default. Bypassing the pagination function, such as by declaring that *cat* is the paginator, would not meet with the intended meaning of this description. However, any ``portable user'' would have to set *PAGER* explicitly to get his or her preferred paginator on all systems. The paginator choice was made partially unspecified, unlike the *VISUAL* editor choice (mandated to be *vi*) because most historical pagers follow a common theme of user input, whereas editors differ dramatically.

Options to specify addresses as **cc** (carbon copy) or **bcc** (blind carbon copy) were considered to be format details and were omitted.

A zero exit status implies that all messages were *sent*, but it gives no assurances that any of them were actually *delivered*. The reliability of the delivery mechanism is unspecified and is an appropriate marketing distinction between systems.

In order to conform to the Utility Syntax Guidelines, a solution was required to the optional *file* option-argument to **–f**. By making *file* an operand, the guidelines are satisfied and users remain portable. However, it does force implementations to support usage such as:

```
mailx -fin mymail.box
```

The **no** *name* method of unsetting variables is not present in all historical systems, but it is in System V and provides a logical set of commands corresponding to the format of the display of options from the *mailx set* command without arguments.

The **ask** and **asksub** variables are the names selected by BSD and System V, respectively, for the same feature. They are synonyms in this volume of POSIX.1-2024.

The *mailx echo* command was not documented in the BSD version and has been omitted here because it is not obviously useful for interactive users.

The default prompt on the System V *mailx* is a <question-mark>, on BSD *Mail* an <ampersand>. Since this volume of POSIX.1-2024 chose the *mailx* name, it kept the System V default, assuming that BSD users would not have difficulty with this minor incompatibility (that they can override).

The meanings of **r** and **R** are reversed between System V *mailx* and SunOS *Mail*. Once again, since this volume of POSIX.1-2024 chose the *mailx* name, it kept the System V default, but allows the SunOS user to achieve the desired results using **flipr**, an internal variable in System V *mailx*, although it has not been documented in the SVID.

The **indentprefix** variable, the **retain** and **unalias** commands, and the **~F** and **~M** command escapes were adopted from 4.3 BSD *Mail*.

The **version** command was not included because no sufficiently general specification of the version information could be devised that would still be useful to a portable user. This command name should be used by suppliers who wish to provide version information about the *mailx* command.

The ``implementation-specific (unspecified) system start-up file'' historically has been named **/etc/mailx.rc**, but this specific name and location are not required.

The intent of the wording for the **next** command is that if any command has already displayed the current message it should display a following message, but, otherwise, it should display the current message. Consider the command sequence:

```
next 3
delete 3
next
```


where the **autoprint** option was not set. The normative text specifies that the second **next** command should display a message following the third message, because even though the current message has not been displayed since it was set by the **delete** command, it has been displayed since the current message was anything other than message number 3. This does not always match historical practice in some implementations, where the command file address followed by **next** (or the default command) would skip the message for which the user had searched.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

Chapter 2 (on page 2472), *ed*, *ls*, *more*, *vi*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 5

The description of the *EDITOR* environment variable is changed to indicate that *ed* is the default editor if this variable is not set. In previous issues, this default was not stated explicitly at this point but was implied further down in the text.

The FUTURE DIRECTIONS section is added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **–F** option is added.
- The **allnet**, **debug**, and **sendwait** internal variables are added.
- The **C**, **ec**, **fo**, **F**, and **S** *mailx* commands are added.

In the DESCRIPTION and ENVIRONMENT VARIABLES sections, text stating ``*HOME* directory'' is replaced by ``directory referred to by the *HOME* environment variable''.

The *mailx* utility is aligned with the IEEE P1003.2b draft standard, which includes various clarifications to resolve IEEE PASC Interpretations submitted for the ISO POSIX-2: 1993 standard. In particular, the changes here address IEEE PASC Interpretations 1003.2 #10, #11, #103, #106, #108, #114, #115, #122, and #129.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/32 is applied, applying a change to the EXTENDED DESCRIPTION, raised by IEEE PASC Interpretation 1003.2 #122, which was overlooked in the first version of this standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/17 is applied, updating the EXTENDED DESCRIPTION (Internal Variables in *mailx*). The system start-up file is changed from ``implementation-defined'' to ``unspecified'' for consistency with other text in the EXTENDED DESCRIPTION.

Issue 7

Austin Group Interpretation 1003.1-2001 #089 is applied, clarifying the effect of the *LC_TIME* environment variable.

Austin Group Interpretation 1003.1-2001 #090 is applied, updating the description of the **next** command.

SD5-XCU-ERN-69 is applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Shading to indicate support for the User Portability Utilities option is added.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0120 [855] and XCU/TC2-2008/0121 [619] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 956 is applied, clarifying how the *PAGER* environment variable and the **crt** internal variable affect pagination.

Austin Group Defects 990 and 991 are applied, changing the description of the **mbox** command.

Austin Group Defect 999 is applied, adding **Save** to the list of commands that are invalid in a start-up file.

Austin Group Defect 1034 is applied, clarifying that **~.** can be used to terminate input mode when **ignoreeof** is set.

Austin Group Defect 1109 is applied, changing the description of the **bang** internal variable.

Austin Group Defect 1113 is applied, changing the description of the *read* state.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1176 is applied, changing ``option-argument'' to ``operand''.

Austin Group Defect 1306 is applied, changing the description of the **folder** internal variable.

Austin Group Defect 1367 is applied, adding the **−E** option.

Austin Group Defect 1401 is applied, changing the requirements for the **reply**, **Reply**, **followup**, and **Followup** commands.

Austin Group Defect 1405 is applied, changing the terminology related to mail messages to match IETF RFC 5322.

Austin Group Defect 1408 is applied, changing the description of Send Mode.

Austin Group Defect 1507 is applied, changing the EXIT STATUS section.

Austin Group Defect 1528 is applied, adding a **"--"** argument to be passed between **"-c"** and *command* when executing shell commands.

Austin Group Defect 1634 is applied, clarifying handling of the system mailbox.

Austin Group Defect 1685 is applied, updating RFC references.

Austin Group Defect 1725 is applied, clarifying that the default for the **screen** internal variable is **noscreen**.

Austin Group Defect 1743 is applied, changing the descriptions of the **metoo** internal variable

and the **alternates** command.

Austin Group Defect 1747 is applied, changing the description of the **alias** command.

105319 ## NAME

105320 make — maintain, update, and regenerate files (**DEVELOPMENT**)

105321 ## SYNOPSIS

105322 SD
```
make [-einpqrst] [-f makefile]... [-j maxjobs] [-k|-S]
    [macro[::[:]]=value...] [target_name...]
```

105324 ## DESCRIPTION

105325 The *make* utility shall update files that are derived from other files. A typical case is one where
105326 object files are derived from the corresponding source files. The *make* utility examines time
105327 relationships and shall update those derived files (called targets) that have modified times
105328 earlier than the modified times of the files (called prerequisites) from which they are derived. A
105329 description file (makefile) contains a description of the relationships between files, and the
105330 commands that need to be executed to update the targets to reflect changes in their
105331 prerequisites. Each specification, or rule, shall consist of a target, optional prerequisites, and
105332 optional commands to be executed when a prerequisite is newer than the target. There are two
105333 kinds of rule:

105334 1. *Inference rules*, which have one target name with at least one <period> (`'.'`) and no
105335 <slash> (`'/'`)

105336 2. *Target rules*, which can have more than one target name

105337 In addition, *make* shall have a collection of built-in macros and inference rules that infer
105338 prerequisite relationships to simplify maintenance of programs.

105339 To receive exactly the behavior described in this section, a portable makefile shall:

105340 • Include the special target **.POSIX**

105341 • Omit any special target reserved for implementations (a leading period followed by
105342 uppercase letters) that has not been specified by this section

105343 The behavior of *make* is unspecified if either or both of these conditions are not met.

105344 ## OPTIONS

105345 The *make* utility shall conform to XBD Section 12.2 (on page 215), except for Guideline 9.

105346 The following options shall be supported:

105347 **−e** Cause environment variables, including those with null values, to override macro
105348 assignments within makefiles.

105349 **−f** *makefile* Specify a different makefile. The argument *makefile* is a pathname of a description
105350 file, which is also referred to as the *makefile*. A pathname of `'-'` shall denote the
105351 standard input. There can be multiple instances of this option, and they shall be
105352 processed in the order specified. The effect of specifying the same option-argument
105353 more than once is unspecified.

105354 **−i** Ignore error codes returned by invoked commands. This mode shall be the same as
105355 if the special target **.IGNORE** were specified without prerequisites.

105356 **−j** *maxjobs* Set the maximum number of targets that can be updated concurrently. If this
105357 option is specified multiple times, the last value of *maxjobs* specified shall take
105358 precedence. If this option is not specified, or if *maxjobs* is 1, only one target shall be
105359 updated at a time (no parallelization). If the value of *maxjobs* is non-positive, the
105360 behavior is unspecified. When *maxjobs* is greater than 1, *make* shall create a pool of
105361 up to *maxjobs* − 1 tokens. (Note that implementations are not required to create a
105362 pool of exactly *maxjobs* − 1 tokens. For example, an implementation could limit the

105363 pool size based on the number of processors available.) If the size of the token pool
105364 would be 0, *make* need not implement a token pool.

105365 When all of the following are true:

105366 • There is a target with commands that is not up-to-date

105367 • The target's prerequisites (if any) are up-to-date

105368 • *make* is not waiting to bring the target up-to-date (see **.WAIT**)

105369 • *make* is currently bringing a different target with commands up-to-date

105370 • *make* is not currently bringing *maxjobs* targets up-to-date in parallel

105371 • The special target **.NOTPARALLEL** is not specified

105372 • The token pool is not empty

105373 then *make* may attempt to remove one token from the pool. If a token is
105374 successfully removed, it shall attempt to bring this target up-to-date in parallel,
105375 and after this processing completes shall return the token to the pool. When *make* is
105376 bringing a target without commands up-to-date, it need not remove a token from
105377 the pool.

105378 If a rule invokes a sub-*make* either via the *MAKE* macro or via a command line that
105379 begins with `'+'`, the sub-*make* is the same implementation as the *make* that invoked
105380 the sub-*make*, and the **–j** option is passed to the sub-*make* via the *MAKEFLAGS*
105381 environment variable with the same *maxjobs* value and is not overridden by a
105382 *maxjobs* value from another source (even if it has the same value), the sub-*make*
105383 shall use the same token pool as its invoking *make* rather than create a new token
105384 pool. Otherwise, it is unspecified whether the sub-*make* uses the same token pool
105385 as its invoking *make* or creates a new token pool. If a rule executes multiple sub-
105386 *make* processes asynchronously the behavior is unspecified.

105387 **–k** Continue to update other targets that do not depend on the current target if a non-
105388 ignored error occurs while executing the commands to bring a target up-to-date.

105389 **–n** Write commands that would be executed on standard output, but do not execute
105390 them. However, lines with a <plus-sign> (`'+'`) prefix, lines that expand the *MAKE*
105391 macro, and lines being processed in order to create an include file or to bring it up-
105392 to-date (see **Include Lines** in the EXTENDED DESCRIPTION section) shall be
105393 executed. In this mode, lines with a <commercial-at> (`'@'`) character prefix shall
105394 be written to standard output.

105395 **–p** Write to standard output the complete set of macro definitions and target
105396 descriptions. The output format is unspecified.

105397 **–q** Return a zero exit value if the target file is up-to-date; otherwise, return an exit
105398 value of 1. Targets shall not be updated if this option is specified. However, a
105399 makefile command line (associated with the targets) with a <plus-sign> (`'+'`)
105400 prefix shall be executed and it is unspecified whether command lines that do not
105401 have a <plus-sign> prefix and either expand the *MAKE* macro or are being
105402 processed in order to create an include file or to bring it up-to-date (see **Include**
105403 **Lines** in the EXTENDED DESCRIPTION section) are executed.

105404 **–r** Clear the suffix list and do not use the built-in rules.

105405 **−S** Terminate *make* if an error occurs while executing the commands to bring a target
105406 up-to-date. This shall be the default and the opposite of **−k**.

105407 **−s** Do not write makefile *execution lines* (see Makefile Execution, on page 3136) or
105408 touch messages (see **−t**) to standard output before executing. This mode shall be
105409 the same as if the special target **.SILENT** were specified without prerequisites.

105410 **−t** Update the modification time of each target as though a *touch target* had been
105411 executed. Targets that have prerequisites but no commands (see Target Rules, on
105412 page 3137), or that are already up-to-date, shall not be touched in this manner.
105413 Write messages to standard output for each target file indicating the name of the
105414 file and that it was touched. Normally, the *makefile* command lines associated with
105415 each target are not executed. However, a command line with a <plus-sign> (`'+'`)
105416 prefix shall be executed and it is unspecified whether command lines that do not
105417 have a <plus-sign> prefix and either expand the *MAKE* macro or are being
105418 processed in order to create an include file or to bring it up-to-date (see **Include**
105419 **Lines** in the EXTENDED DESCRIPTION section) are executed.

105420 Any options specified in the *MAKEFLAGS* environment variable shall be evaluated before any
105421 options specified on the *make* utility command line. If the **−k** and **−S** options are both specified
105422 on the *make* utility command line or by the *MAKEFLAGS* environment variable, the last option
105423 specified shall take precedence. If the **−f** or **−p** options appear in the *MAKEFLAGS* environment
105424 variable, the result is undefined.

105425 **OPERANDS**

105426 The following operands shall be supported:

105427 *target_name* Target names, as defined in the EXTENDED DESCRIPTION section. If no target
105428 is specified, while *make* is processing the makefiles, the first target that *make*
105429 encounters that is not a special target or an inference rule shall be used.

105430 *macro*=*value*
105431 *macro*::=*value*
105432 *macro*:::=*value* Delayed and immediate expansion macro definitions, as defined in Macros (on
105433 page 3139).

105434 Delayed and immediate expansion macro definitions can be intermixed, and shall be processed
105435 in the order specified. If any macro definition appears after a *target_name* operand on the *make*
105436 utility command line, the results are unspecified.

105437 **STDIN**

105438 The standard input shall be used only if the *makefile* option-argument is `'−'`. See the INPUT
105439 FILES section.

105440 **INPUT FILES**

105441 The input file, otherwise known as the makefile, is a text file containing rules, macro definitions,
105442 include lines, and comments. See the EXTENDED DESCRIPTION section.

105443 **ENVIRONMENT VARIABLES**

105444 The following environment variables shall affect the execution of *make*:

105445 *LANG* Provide a default value for the internationalization variables that are unset or null.
105446 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
105447 variables used to determine the values of locale categories.)

105448 *LC_ALL* If set to a non-empty string value, override the values of all the other
105449 internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

MAKEFLAGS
This variable shall be interpreted as a character string representing a series of option characters to be used as the default options. The implementation shall accept both of the following formats (but need not accept them when intermixed):

- The characters are option letters without the leading <hyphen-minus> characters or <blank> separation used on a *make* utility command line.
- The characters are formatted in a manner similar to the use of the *make* utility in shell commands: options are preceded by <hyphen-minus> characters and <blank>-separated as described in XBD Section 12.2 (on page 215). The *macro=value* macro definition operands can also be included. The difference between the contents of *MAKEFLAGS* and the use of the *make* utility in shell commands is that the contents of the variable shall not be subjected to the word expansions (see Section 2.6, on page 2483) associated with parsing shell command lines.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

XSI *PROJECTDIR*
Provide a directory to be used to search for SCCS files not found in the current directory. In all of the following cases, the search for SCCS files is made in the directory **SCCS** in the identified directory. If the value of *PROJECTDIR* begins with a <slash>, it shall be considered an absolute pathname; otherwise, the value of *PROJECTDIR* is treated as a user name and that user's initial working directory shall be examined for a subdirectory **src** or **source**. If such a directory is found, it shall be used. Otherwise, the value is used as a relative pathname.

If *PROJECTDIR* is not set or has a null value, the search for SCCS files shall be made in the directory **SCCS** in the current directory.

The setting of *PROJECTDIR* affects all files listed in the remainder of this utility description for files with a component named **SCCS**.

The value of the *SHELL* environment variable shall not be used as a macro and shall not be modified by defining the *SHELL* macro in a makefile or on the command line. All other environment variables, including those with null values, shall be used as macros, as defined in Macros (on page 3139).

ASYNCHRONOUS EVENTS

For SIGHUP, SIGINT, SIGQUIT, and SIGTERM signals, if the signal was not inherited as ignored, none of the **–n**, **–p**, or **–q** options was specified, *make* is currently processing a target or inference rule, and the current target is neither a directory nor a prerequisite of the special targets **.PHONY** or **.PRECIOUS**:

- The *make* utility shall catch the signal and, if the time of last data modification of the current target has changed since make began processing the rule to bring that target up to date, remove that target; it may also remove that target if the time of last data modification has not changed. Any targets removed in this manner shall be reported in diagnostic or

105496 informational messages of unspecified format, written to standard error.

105497 • If *make* writes a diagnostic message to standard error, it shall exit with a status that
105498 indicates an error occurred; otherwise, it shall set the signal to default and re-signal itself.

105499 In all other circumstances, *make* shall take the standard action for all signals; see Section 1.4 (on
105500 page 2462).

105501 **STDOUT**

105502 If *make* is invoked without any work needing to be done, it may write a message to standard
105503 output indicating that no action was taken. Otherwise, the *make* utility shall write all commands
105504 to be executed (and the filenames of files touched for the **–t** option in a message of unspecified
105505 format) to standard output unless the **–s** option was specified, the command is prefixed with a
105506 <commercial-at> (`'@'`), or the special target **.SILENT** has either the current target as a
105507 prerequisite or has no prerequisites.

105508 **STDERR**

105509 The standard error shall be used for diagnostic messages and may be used for informational
105510 messages about target removals (see ASYNCHRONOUS EVENTS).

105511 **OUTPUT FILES**

105512 Files can be created when the **–t** option is present. Additional files can also be created by the
105513 utilities invoked by *make*.

105514 **EXTENDED DESCRIPTION**

105515 The *make* utility attempts to perform the actions, specified in one or more makefiles, required to
105516 ensure that specified targets are up-to-date. By default, the following files shall be tried in
105517 sequence: **./makefile** and **./Makefile**. If neither **./makefile** nor **./Makefile** is found, other
105518 XSI implementation-defined files may also be tried. On XSI-conformant systems, the additional
105519 files **./s.makefile**, **SCCS/s.makefile**, **./s.Makefile**, and **SCCS/s.Makefile** shall also be tried. The
105520 **–f** option shall direct *make* to ignore any of these default files and use the specified option-
105521 argument as a makefile instead. If this option-argument is `'–'`, standard input shall be used.

105522 The term *makefile* is used to refer to any makefile contents provided by the user, whether in
105523 **./makefile** or its variants, or specified by the **–f** option.

105524 A target shall be considered up-to-date if it exists and is newer than all of its prerequisites, or if it
105525 has already been made up-to-date by the current invocation of *make* (regardless of the target's
105526 existence or age), except that targets that are made up-to-date in order for them to be processed
105527 as include line pathnames (see **Include Lines** below) need not be considered up-to-date during
105528 later processing. A target may also be considered up-to-date if it exists, is the same age as one or
105529 more of its prerequisites, and is newer than the remaining prerequisites (if any). The *make* utility
105530 shall treat all prerequisites as targets themselves and recursively ensure that they are up-to-date,
105531 processing them in the order in which they appear in the rule. The *make* utility shall use the
105532 modification times of files to determine whether the corresponding targets are out-of-date.

105533 To ensure that a target is up-to-date, *make* shall ensure that all of the prerequisites of the target
105534 are up-to-date, then check to see if the target itself is up-to-date. If the target is not up-to-date,
105535 the target shall be made up-to-date by executing the rule's commands (if any). If the target does
105536 not exist after the target has been successfully made up-to-date, the target shall be treated as
105537 being newer than any target for which it is a prerequisite.

105538 If a target exists and there is neither a target rule nor an inference rule for the target, the target
105539 shall be considered up-to-date. It shall be an error if *make* attempts to ensure that a target is up-
105540 to-date but the target does not exist and there is neither a target rule nor an inference rule for the
105541 target.

105542 **Makefile Syntax**

105543 A makefile can contain rules, macro definitions (see Macros, on page 3139), include lines, and
105544 comments. There are two kinds of rules: *target rules*, including special targets (see Target Rules,
105545 on page 3137), and *inference rules* (see Inference Rules, on page 3143). The *make* utility shall
105546 contain a set of built-in inference rules. If the **−r** option is present, the built-in rules shall not be
105547 used and the suffix list shall be cleared. Additional rules of both kinds can be specified in a
105548 makefile. If a rule is defined more than once, the value of the rule shall be that of the last one
105549 specified. Macros can also be defined more than once, and the value of the macro is specified in
105550 Macros (on page 3139). There are three kinds of comments: blank lines, empty lines, and a
105551 <number-sign> (`'#'`) and all following characters up to the first unescaped <newline>
105552 character. Blank lines, empty lines, and lines with <number-sign> (`'#'`) as the first character on
105553 the line are also known as comment lines.

105554 Target and inference rules can contain *command lines*. Command lines can have a prefix that
105555 shall be removed before execution (see Makefile Execution, on page 3136).

105556 When an escaped <newline> (one preceded by a <backslash>) is found anywhere in the
105557 makefile except in a command line after macro expansion, an include line, or a line immediately
105558 preceding an include line, it shall be replaced, along with any leading white space on the next
105559 line, with a single <space>. After all macro expansion is complete, when an escaped <newline>
105560 is found in a command line in a makefile, the command line that is executed shall contain the
105561 <backslash>, the <newline>, and the next line, except that the first character of the next line shall
105562 not be included if it is a <tab>. When an escaped <newline> is found in an include line or in a
105563 line immediately preceding an include line, the behavior is unspecified.

105564 **Include Lines**

105565 If the word **include**, optionally prefixed with a <hyphen-minus> character, appears at the
105566 beginning of a line and is followed by one or more <blank> characters, the string formed by the
105567 remainder of the line shall be processed as follows to produce one or more pathnames:

105568 • The trailing <newline>, any <blank> characters immediately preceding a comment, and
105569 any comment shall be discarded. If the resulting string contains any double-quote
105570 characters (`'"'`) the behavior is unspecified.

105571 • The resulting string shall be processed for macro expansion (see Macros, on page 3139).

105572 • Any <blank> characters that appear after the first non-<blank> shall be used as separators
105573 to divide the macro-expanded string into fields. It is unspecified whether pathname
105574 expansion (see Section 2.14, on page 2523) is also performed.

105575 • If the processing of separators and optional pathname expansion results in zero non-empty
105576 fields, the behavior is unspecified. If it results in at least one non-empty field, these fields
105577 are taken as pathnames.

105578 For each pathname so identified, in the order specified:

105579 • If the pathname does not begin with a `'/'`, it shall be treated as relative to the current
105580 working directory of the process, not relative to the directory containing the makefile.

105581 • The *make* utility shall use one of the following two methods to attempt to create the file or
105582 bring it up-to-date:

105583 1. The ``immediate remaking'' method

105584 If *make* uses this method, any target rules or inference rules for the pathname that
105585 were parsed before the include line was parsed shall be used to attempt to create the

105586 file or to bring it up-to-date before opening the file.

105587 2. The ``delayed remaking'' method

105588 105589 105590 105591 105592 105593 105594 105595 105596 If *make* uses this method, no attempt shall be made to create the file or bring it up-to-date until after the makefile(s) have been read. During processing of the include line, *make* shall read the current contents of the file, if it exists, or treat it as an empty file if it does not exist. Once the makefile(s) have been read, *make* shall use any applicable target rule or inference rule for the pathname, regardless of whether it is parsed before or after the include line, when creating the file or bringing it up-to-date. Additionally in this case, the new contents of the file, if it is successfully created or updated, shall be used when processing rules for the following targets after the makefile(s) have been read:

105597 - The *target_name* operands, if any.
105598 105599 - The first target *make* encounters that is not a special target or an inference rule, if no *target_name* operands are specified.
105600 - All targets that are prerequisites, directly or recursively, of the above targets.

105601 105602 105603 If the pathname is relative, the file does not exist, and an attempt to create it using a rule has not been made and will not be made, it is unspecified whether additional directories are searched for an existing file of the same relative pathname.

105604 If, after proceeding as described above, the file still cannot be opened:

105605 105606 - If the word **include** was prefixed with a <hyphen-minus> character, the file shall be ignored.
105607 - Otherwise, an error shall occur.

105608 105609 105610 - The contents of the file specified by the pathname shall be read and processed as if they appeared in the makefile in place of the include line. If the file ends with an escaped <newline> the behavior is unspecified.
105611 105612 - The file may itself contain further include lines. Implementations shall support nesting of include files up to a depth of at least 16.

105613 **Makefile Execution**

105614 Makefile command lines shall be processed one at a time.

105615 105616 105617 Makefile command lines can have one or more of the following prefixes: a <hyphen-minus> (`'-'`), a <commercial-at> (`'@'`), or a <plus-sign> (`'+'`). These shall modify the way in which *make* processes the command.

105618 105619 105620 – If the command prefix contains a <hyphen-minus>, or the **−i** option is present, or the special target **.IGNORE** has either the current target as a prerequisite or has no prerequisites, any error found while executing the command shall be ignored.

105621 105622 105623 105624 `@` If the command prefix contains a <commercial-at> and the *make* utility command line **−n** option is not specified, or the **−s** option is present, or the special target **.SILENT** has either the current target as a prerequisite or has no prerequisites, the command shall not be written to standard output before it is executed.

105625 105626 + If the command prefix contains a <plus-sign>, the command shall be executed even if **−n**, **−q**, or **−t** is specified.

105627 An *execution line* is built from the command line by removing any prefix characters. Except as

105628 described under the <commercial-at> (`'@'`) prefix, the execution line shall be written to the
105629 standard output, optionally preceded by a <tab>. The execution line shall then be executed by a
105630 shell as if it were passed as the argument to the *system*() interface, except that if errors are not
105631 being ignored then the shell **−e** option shall also be in effect. If errors are being ignored for the
105632 command (as a result of the **−i** option, a `'−'` command prefix, or a **.IGNORE** special target), the
105633 shell **−e** option shall not be in effect. The environment for the command being executed shall
105634 contain all of the variables in the environment of *make*.

105635 By default, when *make* receives a non-zero status from the execution of a command, it shall
105636 terminate with an error message to standard error.

105637 **Target Rules**

105638 Target rules are formatted as follows:

105639 *target* **[***target***...]: [***prerequisite***...][;***command***]**
105640 **[**<tab>*command*
105641 <tab>*command*
105642 **...]**

105643 Target entries are specified by a <blank>-separated, non-null list of targets, then a <colon>, then
105644 a <blank>-separated, possibly empty list of prerequisites. Text following a <semicolon>, if any,
105645 and all following lines that begin with a <tab>, are makefile command lines to be executed to
105646 update the target. The first non-empty line that does not begin with a <tab> or `'#'` shall begin a
105647 new entry. Any comment line may begin a new entry.

105648 Applications shall select target names from the set of characters consisting solely of slashes,
105649 hyphens, periods, underscores, digits, and alphabetics from the portable character set (see XBD
105650 Section 6.1, on page 117). Implementations may allow other characters in target names as
105651 extensions. The interpretation of targets containing the characters `'%'` and `'"'` is
105652 implementation-defined.

105653 A target that has prerequisites, but does not have any commands, can be used to add to the
105654 prerequisite list for that target. Only one target rule for any given target can contain commands.

105655 Lines that begin with one of the following are called *special targets* and control the operation of
105656 *make*:

105657 **.DEFAULT** If the makefile contains this special target, the application shall ensure that it is
105658 specified with commands, but without prerequisites. The commands shall be used
105659 by *make* if there are no other rules available to build a target.

105660 **.IGNORE** Prerequisites of this special target are targets themselves; this shall cause errors
105661 from commands associated with them to be ignored in the same manner as
105662 specified by the **−i** option. Subsequent occurrences of **.IGNORE** shall add to the
105663 list of targets ignoring command errors. If no prerequisites are specified, *make* shall
105664 behave as if the **−i** option had been specified and errors from all commands
105665 associated with all targets shall be ignored.

105666 **.NOTPARALLEL**
105667 The application shall ensure that this special target is specified without
105668 prerequisites or commands. When specified, *make* shall update one target at a time,
105669 regardless of whether the **−j** *maxjobs* option is specified. If the **−j** *maxjobs* option is
105670 specified, the option shall continue to be passed unchanged to sub-*make*
105671 invocations via *MAKEFLAGS*.

105672 **.PHONY** Prerequisites of this special target are targets themselves; these targets (known as
105673 *phony targets*) shall be considered always out-of-date when the *make* utility begins
105674 executing. If a phony target's commands are executed, that phony target shall then
105675 be considered up-to-date until the execution of *make* completes. Subsequent
105676 occurrences of **.PHONY** shall add to the list of phony targets. A **.PHONY** special
105677 target with no prerequisites shall be ignored. If the **−t** option is specified, phony
105678 targets shall not be touched. Phony targets shall not be removed if *make* receives
105679 one of the asynchronous events explicitly described in the ASYNCHRONOUS
105680 EVENTS section.

105681 **.POSIX** The application shall ensure that this special target is specified without
105682 prerequisites or commands. If it appears as the first non-comment line in the
105683 makefile, *make* shall process the makefile as specified by this section; otherwise, the
105684 behavior of *make* is unspecified.

105685 **.PRECIOUS** Prerequisites of this special target shall not be removed if *make* receives one of the
105686 asynchronous events explicitly described in the ASYNCHRONOUS EVENTS
105687 section. Subsequent occurrences of **.PRECIOUS** shall add to the list of precious
105688 files. If no prerequisites are specified, all targets in the makefile shall be treated as
105689 if specified with **.PRECIOUS**.

105690 XSI **.SCCS_GET** The application shall ensure that this special target is specified without
105691 prerequisites. If this special target is included in a makefile, the commands
105692 specified with this target shall replace the default commands associated with this
105693 special target (see Default Rules, on page 3146). The commands specified with this
105694 target are used to get all SCCS files that are not found in the current directory.

105695 When source files are named in a list of prerequisites, *make* shall treat them just like
105696 any other target. Because the source file is presumed to be present in the directory,
105697 there is no need to add an entry for it to the makefile. When a target has no
105698 prerequisites, but is present in the directory, *make* shall assume that that file is up-
105699 to-date. If, however, an SCCS file named **SCCS/s.***source_file* is found for a target
105700 *source_file*, *make* compares the timestamp of the target file with that of the
105701 **SCCS/s.source_file** to ensure the target is up-to-date. If the target is missing, or if
105702 the SCCS file is newer, *make* shall automatically issue the commands specified for
105703 the **.SCCS_GET** special target to retrieve the most recent version. However, if the
105704 target is writable by anyone, *make* shall not retrieve a new version.

105705 **.SILENT** Prerequisites of this special target are targets themselves; this shall cause
105706 commands associated with them not to be written to the standard output before
105707 they are executed. Subsequent occurrences of **.SILENT** shall add to the list of
105708 targets with silent commands. If no prerequisites are specified, *make* shall behave
105709 as if the **−s** option had been specified and no commands or touch messages
105710 associated with any target shall be written to standard output.

105711 **.SUFFIXES** Prerequisites of **.SUFFIXES** shall be appended to the list of known suffixes and are
105712 used in conjunction with the inference rules (see Inference Rules, on page 3143). If
105713 **.SUFFIXES** does not have any prerequisites, the list of known suffixes shall be
105714 cleared.

105715 **.WAIT** The application shall ensure that this special target, if specified as a target, is
105716 specified without prerequisites or commands. When **.WAIT** appears as a target, it
105717 shall have no effect. When **.WAIT** appears in a target rule as a prerequisite, it shall
105718 not itself be treated as a prerequisite; however, *make* shall not recursively process
105719 the prerequisites (if any) to the right of the **.WAIT** until the prerequisites (if any) to

105720 the left of it have been brought up-to-date. Implementations may also enforce the
105721 same ordering between the affected prerequisites while processing other target
105722 rules that have some or all of the same affected prerequisites.

105723 The special targets **.IGNORE**, **.NOTPARALLEL**, **.PHONY**, **.POSIX**, **.PRECIOUS**, **.SILENT**,
105724 **.SUFFIXES**, and **.WAIT** shall be specified without commands.

105725 Targets and prerequisites consisting of a leading <period> followed by the uppercase letters
105726 "POSIX" and then any other characters are reserved for future standardization. Targets and
105727 prerequisites consisting of a leading <period> followed by one or more uppercase letters, that
105728 are not described above, are reserved for implementation extensions.

105729 **Macros**

105730 A macro can be one of two flavors, *delayed-expansion* or *immediate-expansion*.

105731 The following form defines a delayed-expansion macro (replacing any previous definition of the
105732 macro named by *string1*):

105733 *string1* = **[***string2***]**

105734 The following form defines an immediate-expansion macro (replacing any previous definition of
105735 the macro named by *string1*):

105736 *string1* ::= **[***string2***]**

105737 The following form defines a delayed-expansion macro (replacing any previous definition of the
105738 macro named by *string1*):

105739 *string1* :::= **[***string2***]**

105740 by immediately expanding macros in *string2*, if any, before assigning the value.

105741 The following form defines a delayed-expansion macro (replacing any previous definition of the
105742 macro named by *string1*):

105743 *string1* != **[***string2***]**

105744 by immediately expanding macros in *string2*, if any, and then executing the result as a shell
105745 command as if it were passed as the argument to the *system*( ) interface. The *make* utility shall
105746 capture the standard output from the shell execution and shall remove all white space at the
105747 beginning, remove a single trailing <newline> character (if there is one), and then replace all
105748 remaining <newline> characters with <space> characters to produce the value assigned to the
105749 macro named by *string1*. It shall not be an error if the shell command has non-zero exit status.

105750 The following form defines a delayed-expansion macro, but only if the macro named by *string1*
105751 is not already defined:

105752 *string1* ?= **[***string2***]**

105753 The following form (the *append* form) appends additional text to the value of a macro:

105754 *string1* += **[***string2***]**

105755 When using the append form:

105756 • If the macro named by *string1* does not exist, this form shall be equivalent to the delayed-
105757 expansion form

105758 *string1* = **[***string2***]**

105759 • If the macro named by *string1* exists and is an immediate-expansion macro, then a <space>
105760 or <tab> character followed by the evaluation of *string2* shall be appended to the value
105761 currently assigned to the macro named by *string1*.

105762 • If the macro named by *string1* exists and is a delayed-expansion macro, then a <space> or
105763 <tab> character followed by the unevaluated *string2* shall be appended to the value
105764 currently assigned to the macro named by *string1*.

105765 In all cases the value of *string1* is defined as all characters from the first non-<blank> character to
105766 the last non-<blank> character, inclusive, before the =, `::=`, `:::=`, `!=`, `?=`, or +=. Portable
105767 applications shall ensure that a <blank> precedes the `::=`, `:::=`, `!=`, `?=`, or += in those forms to
105768 avoid any parsing ambiguity with implementations that permit <colon>, <exclamation-mark>,
105769 <question-mark>, or <plus-sign> in macro names as extensions. The value of *string2* is defined
105770 as all characters from the first non-<blank> character, if any, after the <equals-sign>, up to but
105771 not including a comment character (`'#'`) or an unescaped <newline>.

105772 Portable applications shall select macro names from the set of characters consisting solely of
105773 characters from the portable filename character set. Implementations may allow other characters
105774 in macro names as extensions; however, a macro name shall not contain an <equals-sign>,
105775 <blank>, or control character.

105776 Macro expansions in *string1* of macro definition lines shall be evaluated when read. Macro
105777 expansions in *string2* of macro definition lines shall be performed according to the form of
105778 macro definition used. In immediate-expansion forms (including appending to an existing
105779 immediate-expansion macro), they shall be expanded in the macro definition line and the result
105780 of the expansion shall not be scanned for further macros when the macro identified by *string1* is
105781 expanded. In delayed-expansion forms (including appending to an existing delayed-expansion
105782 macro, and conditional assignment to a macro not previously existing), they shall not be
105783 expanded in the macro definition line; they shall be expanded when the macro identified by
105784 *string1* is expanded, and the result of the expansion shall be scanned for further macros.
105785 Implementations shall support at least 100 levels of indirection.

105786 Macros can appear anywhere in the makefile. Macro expansions using the forms $(*string1*) or
105787 ${*string1*} shall be replaced by *string2*, as follows:

105788 • Macros in target lines shall be evaluated when the target line is read.

105789 • Macros in makefile command lines shall be evaluated when the command is executed.

105790 • Macros in the string before the <equals-sign> in a macro definition shall be evaluated
105791 when the macro assignment is made.

105792 • Immediate-expansion macros shall be evaluated immediately when the macro assignment
105793 is made, and this value shall be used as the replacement until the immediate-expansion
105794 macro is redefined.

105795 • Delayed-expansion macros after the <equals-sign> in macro definitions other than the
105796 `:::=`, `!=`, and += forms, and after the <equals-sign> in += form macro definitions where
105797 the macro named by *string1* exists and is a delayed-expansion macro, shall only be
105798 evaluated when the defined macro is expanded.

105799 The parentheses or braces are optional if *string1* is a single character. The string `"$$"` shall be
105800 replaced by the single character `'$'`, except during the immediate expansion performed for the
105801 `:::=` operator, where it shall be left unmodified. If *string1* in a macro expansion contains a
105802 macro expansion, that inner macro expansion shall be performed first and the result substituted
105803 into *string1* to produce the macro name used for the outer macro expansion.

105804 Macro expansions using the forms $(*string1*:*subst1*=[*subst2*]) or ${*string1*:*subst1*=[*subst2*]} can be

105805 used to replace all occurrences of *subst1* with *subst2* when the macro substitution is performed.
105806 The *subst1* to be replaced shall be recognized when it is a suffix at the end of a word in *string1*
105807 (where a *word*, in this context, is defined to be a string delimited by the beginning of the value, a
105808 <blank>, or a <newline>). If *string1* in a macro expansion contains a macro expansion, that inner
105809 macro expansion shall be performed as described above and the result substituted into *string1* to
105810 produce the macro name used for the outer macro expansion.

105811 Macro expansions using the forms $(*string1*:[*op*]%[*os*]=[*np*][%][*ns*]) or
105812 ${*string1*:[*op*]%[*os*]=[*np*][%][*ns*]} are called pattern macro expansions, where *op* is the old prefix,
105813 *os* is the old suffix, *np* is the new prefix and *ns* is the new suffix. Any item inside square brackets
105814 is optional. With this form, when the macro *string1* is expanded each white-space-separated
105815 word that completely matches the [*op*]%[*os*] pattern on the left-hand side of the <equals-sign>
105816 (`'='`), where the <percent> (`'%'`) character matches zero or more characters, shall be replaced
105817 by the right-hand side of the <equals-sign> and shall then be further modified according to the
105818 use of <percent> characters as described below. Any words that do not match shall be
105819 unmodified in the expansion.

105820 If more than one <percent> character appears on the left-hand side of the <equals-sign> (`'='`),
105821 the second and subsequent <percent> characters shall be treated as literal characters in *os*.

105822 If no <percent> character appears on the right-hand side of the <equals-sign>, no further
105823 modification of the word shall be performed. If a single <percent> character appears on the
105824 right-hand side, the <percent> character in the word shall be replaced with the characters
105825 matched by the <percent> on the left-hand side. If more than one <percent> character appears
105826 on the right-hand side, it is unspecified whether the first <percent> character in the word is
105827 replaced with the characters matched by the <percent> on the left-hand side and all remaining
105828 <percent> characters are left unchanged, or each <percent> character is replaced with the
105829 characters matched by the <percent> on the left-hand side.

105830 In both macro expansion forms, any macro expansions on the right-hand side of the <colon>
105831 shall be recursively expanded before further examination. If this results in more than one
105832 <equals-sign> after the <colon>, the first one shall be the separator.

105833 In all forms of macro expansion, if the value of the macro named by *string1* is an empty string, or
105834 if the macro named by *string1* does not exist, the final result shall be an empty string.

105835 **Note:** It is not safe to assume that a macro which has not intentionally been set to a specific value will
105836 not exist. See APPLICATION USAGE for more information.

105837 Macro definitions shall be taken from the following sources, in the following logical order,
105838 before the makefile(s) are read.

105839 1. Macros specified on the *make* utility command line, in the order specified on the
105840 command line. It is unspecified whether the internal macros defined in Internal Macros
105841 (on page 3144) are accepted from this source.

105842 2. Macros defined by the *MAKEFLAGS* environment variable, in the order specified in the
105843 environment variable. It is unspecified whether the internal macros defined in Internal
105844 Macros (on page 3144) are accepted from this source.

105845 3. The contents of the environment, excluding the *MAKEFLAGS* and *SHELL* variables and
105846 including the variables with null values.

105847 4. Macros defined in the inference rules built into *make*.

105848 Macro definitions from these sources shall not override macro definitions from a lower-
105849 numbered source. Macro definitions from a single source (for example, the *make* utility
105850 command line, the *MAKEFLAGS* environment variable, or the other environment variables)

105851 shall override previous macro definitions from the same source.

105852 Macros defined in the makefile(s) shall override macro definitions that occur before them in the
105853 makefile(s) and macro definitions from source 4. If the **−e** option is not specified, macros defined
105854 in the makefile(s) shall override macro definitions from source 3. Macros defined in the
105855 makefile(s) shall not override macro definitions from source 1 or source 2.

105856 Before the makefile(s) are read, all of the *make* utility command line options (except **−f** and **−p**)
105857 and *make* utility command line macro definitions (except any for the *MAKEFLAGS* macro), not
105858 already included in the *MAKEFLAGS* macro, shall be added to the *MAKEFLAGS* macro, quoted
105859 in an implementation-defined manner such that when *MAKEFLAGS* is read by another instance
105860 of the *make* command, the original macro's value is recovered. Other implementation-defined
105861 options and macros, with the exception of the *CURDIR* macro, may also be added to the
105862 *MAKEFLAGS* macro. If this modifies the value of the *MAKEFLAGS* macro, or, if the
105863 *MAKEFLAGS* macro is modified at any subsequent time, the *MAKEFLAGS* environment variable
105864 shall be modified to match the new value of the *MAKEFLAGS* macro. The result of setting
105865 *MAKEFLAGS* in the Makefile is unspecified.

105866 Before the makefile(s) are read, all of the *make* utility command line macro definitions (except the
105867 *MAKEFLAGS* macro or the *SHELL* macro) shall be added to the environment of *make*. Other
105868 implementation-defined variables may also be added to the environment of *make*. Macros
105869 defined by the *MAKEFLAGS* environment variable and macros defined in the makefile(s) shall
105870 not be added to the environment of *make* if they are not already in its environment. With the
105871 exception of *SHELL* (see below), it is unspecified whether macros defined in these ways update
105872 the value of an environment variable that already exists in the environment of *make*.

105873 The *MAKE* macro shall be treated specially. If *MAKE* is not defined in the environment, the
105874 *MAKE* macro shall be provided by *make* and set to the value of *argv*[0] passed to *main*( ) (or
105875 equivalent, if *make* is not a C program). If this value contains at least one <slash> and is a relative
105876 pathname, *make* shall convert it to an absolute pathname. If *MAKE* is defined in the makefile or
105877 is specified on the command line, it shall replace the original value of the *MAKE* macro.

105878 The *SHELL* macro shall be treated specially. It shall be provided by *make* and set to the pathname
105879 of the shell command language interpreter (see *sh*). The *SHELL* environment variable shall not
105880 affect the value of the *SHELL* macro. If *SHELL* is defined in the makefile or is specified on the
105881 command line, it shall replace the original value of the *SHELL* macro, but shall not affect the
105882 *SHELL* environment variable. Other effects of defining *SHELL* in the makefile or on the
105883 command line are implementation-defined.

105884 The *CURDIR* macro shall be treated specially. It shall be provided by *make* and set to an absolute
105885 pathname of the current working directory when *make* is executed. The value shall be the same
105886 as the pathname that would be output by the *pwd* utility with either the **−L** or **−P** option; if they
105887 differ, it is unspecified which value is used. The *CURDIR* environment variable shall not affect
105888 the value of the *CURDIR* macro unless the **−e** option is specified. If the **−e** option is not specified,
105889 there is a *CURDIR* environment variable set, and its value is different from the *CURDIR* macro
105890 value, the environment variable value shall be set to the macro value. If *CURDIR* is defined in
105891 the makefile, present in the *MAKEFLAGS* environment variable, or specified on the command
105892 line, it shall replace the original value of the *CURDIR* macro in accordance with the logical order
105893 described above, but shall not cause *make* to change its current working directory.

105894 **Inference Rules**

105895 Inference rules are formatted as follows:

105896 *target*:
105897 <tab>*command*
105898 [<tab>*command*]
105899 ...

105900 *line that does not begin with* <tab> *or* #

105901 The application shall ensure that the *target* portion is a valid target name (see Target Rules, on
105902 page 3137) of the form ***.s2*** or ***.s1.s2*** (where ***.s1*** and ***.s2*** are suffixes that have been given as
105903 prerequisites of the **.SUFFIXES** special target and ***s1*** and ***s2*** do not contain any <slash> or
105904 <period> characters.) If there is only one <period> in the target, it is a single-suffix inference
105905 rule. Targets with two periods are double-suffix inference rules. Inference rules can have only
105906 one target before the <colon>.

105907 The application shall ensure that the makefile does not specify prerequisites for inference rules;
105908 no characters other than white space shall follow the <colon> in the first line, except when
105909 creating the *empty rule,* described below. Prerequisites are inferred, as described below.

105910 Inference rules can be redefined. A target that matches an existing inference rule shall overwrite
105911 the old inference rule. An empty rule can be created with a command consisting of simply a
105912 <semicolon> (that is, the rule still exists and is found during inference rule search, but since it is
105913 empty, execution has no effect). The empty rule can also be formatted as follows:

105914 *rule*: ;

105915 where zero or more <blank> characters separate the <colon> and <semicolon>.

105916 The *make* utility uses the suffixes of targets and their prerequisites to infer how a target can be
105917 made up-to-date. A list of inference rules defines the commands to be executed. By default, *make*
105918 contains a built-in set of inference rules. Additional rules can be specified in the makefile.

105919 The special target **.SUFFIXES** contains as its prerequisites a list of suffixes that shall be used by
105920 the inference rules. The order in which the suffixes are specified defines the order in which the
105921 inference rules for the suffixes are used. New suffixes shall be appended to the current list by
105922 specifying a **.SUFFIXES** special target in the makefile. A **.SUFFIXES** target with no prerequisites
105923 shall clear the list of suffixes. An empty **.SUFFIXES** target followed by a new **.SUFFIXES** list is
105924 required to change the order of the suffixes.

105925 Normally, the user would provide an inference rule for each suffix. The inference rule to update
105926 a target with a suffix ***.s1*** from a prerequisite with a suffix ***.s2*** is specified as a target ***.s2.s1***. The
105927 internal macros provide the means to specify general inference rules (see Internal Macros, on
105928 page 3144).

105929 When no target rule with commands is found to update a target, the inference rules shall be
105930 checked. The suffix of the target (***.s1***) to be built shall be compared to the list of suffixes specified
105931 by the **.SUFFIXES** special targets. If the ***.s1*** suffix is found in **.SUFFIXES**, the inference rules
105932 shall be searched in the order defined for the first ***.s2.s1*** rule whose prerequisite file (**$***.*s2***) exists.
105933 If the target is out-of-date with respect to this prerequisite, the commands for that inference rule
105934 shall be executed. Prerequisites added by target rules without commands shall not affect the
105935 selection of the applicable inference rule.

105936 If the target to be built does not contain a suffix and there is no rule for the target, the single-
105937 suffix inference rules shall be checked. The single-suffix inference rules define how to build a
105938 target if a file is found with a name that matches the target name with one of the single suffixes

appended. A rule with one suffix ***.s2*** is the definition of how to build ***target*** from ***target.s2***. The other suffix (***.s1***) is treated as null.

XSI A <tilde> (`'~'`) in the above rules refers to an SCCS file in the current directory. Thus, the rule **.c˜.o** would transform an SCCS C-language source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix, it is incompatible with *make*'s suffix point of view. Hence, the `'~'` is a way of changing any file reference into an SCCS file reference.

Libraries

If a target or prerequisite contains parentheses, it shall be treated as a member of an archive library. For the *lib*(*member***.o**) expression *lib* refers to the name of the archive library and *member***.o** to the member name. The application shall ensure that the member is an object file with the **.o** suffix. The modification time of the expression is the modification time for the member as kept in the archive library; see *ar*. The **.a** suffix shall refer to an archive library. The ***.s2*.a** rule shall be used to update a member in the library from a file with a suffix ***.s2***.

Internal Macros

The *make* utility shall maintain a set of internal macros that can be used in the commands of target and inference rules, as described below. In order to clearly define the meaning of these macros, some clarification of the terms *target rule*, *inference rule*, *target*, and *prerequisite* is necessary.

Target rules are specified by the user in a makefile for a particular target. Inference rules are user-specified or *make*-specified rules for a particular class of target name. Explicit prerequisites are those prerequisites specified in a makefile on target lines. Implicit prerequisites are those prerequisites that are generated when inference rules are used. Inference rules are applied to implicit prerequisites or to explicit prerequisites that do not have target rules defined for them in the makefile. Target rules are applied to targets specified in the makefile.

Before any target in the makefile is updated, each of its prerequisites (both explicit and implicit) shall be updated. This shall be accomplished by recursively processing each prerequisite. Upon recursion, each prerequisite shall become a target itself. Its prerequisites in turn shall be processed recursively until a target is found that has no prerequisites, or further recursion would require applying two inference rules one immediately after the other, at which point the recursion shall stop. As an extension, implementations may continue recursion when two or more successive inference rules need to be applied; however, if there are multiple different chains of such rules that could be used to create the target, it is unspecified which chain is used. The recursion shall then back up, updating each target as it goes.

In the definitions that follow, the word *target* refers to one of:

- A target specified in the makefile
- An explicit prerequisite specified in the makefile that becomes the target when *make* processes it during recursion
- An implicit prerequisite that becomes a target when *make* processes it during recursion

In the definitions that follow, the word *prerequisite* refers to one of:

- An explicit prerequisite specified in the makefile for a particular target
- An implicit prerequisite generated as a result of locating an appropriate inference rule and corresponding file that matches the suffix of the target

The internal macros are:

105982 $@ The $@ macro shall evaluate to the full target name of the current target, or the archive
105983 filename part of a library archive target. It shall be evaluated for both target and
105984 inference rules.

105985 For example, in the **.c.a** inference rule, $@ represents the out-of-date **.a** file to be built.
105986 Similarly, in a makefile target rule to build **lib.a** from **file.c**, $@ represents the out-of-
105987 date **lib.a**.

105988 $% The $% macro shall be evaluated only when the current target is an archive library
105989 member of the form *libname*(*member***.o**). In these cases, $@ shall evaluate to *libname* and
105990 $% shall evaluate to *member***.o**. The $% macro shall be evaluated for both target and
105991 inference rules.

105992 For example, in a makefile target rule to build **lib.a**(**file.o**), $% represents **file.o**, as
105993 opposed to $@, which represents **lib.a**.

105994 $^ The $^ macro shall evaluate to the list of prerequisites for the current target, with any
105995 duplicates (except the first) removed. It shall be evaluated for both target and inference
105996 rules. If the list of prerequisites of the target contains any **.WAIT** special targets, the
105997 results of expanding $^ are unspecified.

105998 For example, in a makefile target rule to build *prog* from **file1.o**, **file2.o**, and **file3.o**, and
105999 regardless of which prerequisites *prog* is out-of-date with respect to, $^ represents
106000 **file1.o**, **file2.o**, and **file3.o**.

106001 $+ The $+ macro shall be equivalent to $^, except that duplicates shall not be removed; all
106002 prerequisites shall appear in the order they were listed in the makefile.

106003 $? The $? macro shall evaluate to the list of prerequisites that are newer than the current
106004 target. It shall be evaluated for both target and inference rules.

106005 For example, in a makefile target rule to build *prog* from **file1.o**, **file2.o**, and **file3.o**, and
106006 where *prog* is not out-of-date with respect to **file1.o**, but is out-of-date with respect to
106007 **file2.o** and **file3.o**, $? represents **file2.o** and **file3.o**.

106008 $< In an inference rule, the $< macro shall evaluate to the filename whose existence
106009 allowed the inference rule to be chosen for the target. In the **.DEFAULT** rule, the $<
106010 macro shall evaluate to the current target name. The meaning of the $< macro is
106011 otherwise unspecified.

106012 For example, in the **.c.a** inference rule, $< represents the prerequisite **.c** file.

106013 $* The $* macro shall evaluate to the current target name with its suffix deleted. It shall be
106014 evaluated at least for inference rules.

106015 For example, in the **.c.a** inference rule, $*.o represents the out-of-date **.o** file that
106016 corresponds to the prerequisite **.c** file.

106017 Each of the internal macros has an alternative form. When an uppercase `'D'` or `'F'` is appended
106018 to any of the macros, the meaning shall be changed to the *directory part* for `'D'` and *filename part*
106019 for `'F'`. The directory part is the path prefix of the file without a trailing <slash>; for the current
106020 directory, the directory part is `'.'`. When the $? macro contains more than one prerequisite
106021 filename, the $(?D) and $(?F) (or ${?D} and ${?F}) macros expand to a list of directory name parts
106022 and filename parts respectively.

106023 For the target *lib*(*member***.o**) and the ***.s2*.a** rule, the internal macros shall be defined as:

106024 $< *member**.s2***
106025 $* *member*
106026 $@ *lib*
106027 $^ *member**.s2***
106028 $? *member**.s2***
106029 $% *member***.o**

106030 **Default Rules**

106031 The default rules and macro values for *make* shall achieve results that are the same as if the
106032 following were used, except that where a result includes the literal value of a macro, this value
106033 may differ. Implementations that do not support the C-Language Development Utilities option
106034 may omit **CC**, **CFLAGS**, **YACC**, **YFLAGS**, **LEX**, **LFLAGS**, **LDFLAGS**, and the **.c**, **.y**, and **.l**
106035 inference rules. Implementations may provide additional macros and rules.

106036 *SPECIAL TARGETS*

106037 XSI
```
.SCCS_GET:
    sccs $(SCCSFLAGS) get $(SCCSGETFLAGS) $@
```

106039 XSI
```
.SUFFIXES: .o .c .y .l .a .sh .c~ .y~ .l~ .sh~
```

106040 *MACROS*

```
AR=ar
ARFLAGS=-rv
YACC=yacc
YFLAGS=
LEX=lex
LFLAGS=
LDFLAGS=
CC=c17
CFLAGS=-O 1
```
106050 XSI
```
GET=get
GFLAGS=
SCCSFLAGS=
SCCSGETFLAGS=-s
```

106054 *SINGLE-SUFFIX RULES*

```
.c:
    $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $<
.sh:
    cp $< $@
    chmod a+x $@
```
106060 XSI
```
.c~:
    $(GET) $(GFLAGS) -p $< > $*.c
    $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $*.c
.sh~:
    $(GET) $(GFLAGS) -p $< > $*.sh
```



```
106065          cp $*.sh $@
106066          chmod a+x $@
```

106067 *DOUBLE-SUFFIX RULES*

```
106068          .c.o:
106069              $(CC) $(CFLAGS) -c $<
106070          .y.o:
106071              $(YACC) $(YFLAGS) $<
106072              $(CC) $(CFLAGS) -c y.tab.c
106073              rm -f y.tab.c
106074              mv y.tab.o $@
106075          .l.o:
106076              $(LEX) $(LFLAGS) $<
106077              $(CC) $(CFLAGS) -c lex.yy.c
106078              rm -f lex.yy.c
106079              mv lex.yy.o $@
106080          .y.c:
106081              $(YACC) $(YFLAGS) $<
106082              mv y.tab.c $@
106083          .l.c:
106084              $(LEX) $(LFLAGS) $<
106085              mv lex.yy.c $@
```

106086 XSI

```
106086          .c~.o:
106087              $(GET) $(GFLAGS) -p $< > $*.c
106088              $(CC) $(CFLAGS) -c $*.c
106089          .y~.o:
106090              $(GET) $(GFLAGS) -p $< > $*.y
106091              $(YACC) $(YFLAGS) $*.y
106092              $(CC) $(CFLAGS) -c y.tab.c
106093              rm -f y.tab.c
106094              mv y.tab.o $@
106095          .l~.o:
106096              $(GET) $(GFLAGS) -p $< > $*.l
106097              $(LEX) $(LFLAGS) $*.l
106098              $(CC) $(CFLAGS) -c lex.yy.c
106099              rm -f lex.yy.c
106100              mv lex.yy.o $@
106101          .y~.c:
106102              $(GET) $(GFLAGS) -p $< > $*.y
106103              $(YACC) $(YFLAGS) $*.y
106104              mv y.tab.c $@
106105          .l~.c:
106106              $(GET) $(GFLAGS) -p $< > $*.l
106107              $(LEX) $(LFLAGS) $*.l
106108              mv lex.yy.c $@
```



```
.c.a:
    $(CC) -c $(CFLAGS) $<
    $(AR) $(ARFLAGS) $@ $*.o
    rm -f $*.o
```

EXIT STATUS

When the **-q** option is specified, the *make* utility shall exit with one of the following values:

0 All specified targets were already up-to-date.

1 One or more targets were not up-to-date.

>1 An error occurred.

When the **-q** option is not specified, the *make* utility shall exit with one of the following values:

0 All specified targets were already up-to-date, or all commands executed to bring targets up-to-date either exited with status 0 or had a non-zero exit status that was specified (via the **-i** option, the special target **.IGNORE**, or a `'-'` command prefix) to be ignored.

>0 An error occurred, or at least one command executed to bring a target up-to-date exited with a non-zero exit status that was not specified to be ignored.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

If there is a source file (such as **./source.c**) and there are two SCCS files corresponding to it (**./s.source.c** and **./SCCS/s.source.c**), on XSI-conformant systems *make* uses the SCCS file in the current directory. However, users are advised to use the underlying SCCS utilities (*admin*, *delta*, *get*, and so on) or the *sccs* utility for all source files in a given directory. If both forms are used for a given source file, future developers are very likely to be confused.

It is incumbent upon portable makefiles to specify the **.POSIX** special target in order to guarantee that they are not affected by local extensions.

The **-k** and **-S** options are both present so that the relationship between the command line, the *MAKEFLAGS* variable, and the makefile can be controlled precisely. If the **k** flag is passed in *MAKEFLAGS* and a command is of the form:

```
$(MAKE) -S foo
```

then the default behavior is restored for the child *make*.

When the **-n** option is specified, it is always added to *MAKEFLAGS*. This allows a recursive *make* **-n** *target* to be used to see all of the actions that would be taken to update *target*.

Because of widespread historical practice, interpreting a <number-sign> (`'#'`) inside a variable as the start of a comment has the unfortunate side-effect of making it impossible to place a <number-sign> in a variable, thus forbidding something like:

```
CFLAGS = -D "COMMENT_CHAR='#'"
```

Many historical *make* utilities stop chaining together inference rules when an intermediate target is nonexistent. For example, it might be possible for a *make* to determine that both **.y.c** and **.c.o** could be used to convert a **.y** to a **.o**. Instead, in this case, *make* requires the use of a **.y.o** rule.

The standard set of default rules uses only features provided by other parts of this volume of POSIX.1-2024. They include rules for optional utilities in this volume of POSIX.1-2024, but only rules pertaining to utilities that are provided are needed in an implementation's default set.

Although *make* expands macros that do not exist to an empty string, it is not safe to assume that a macro which has not intentionally been set to a specific value will expand to an empty string for everyone who uses the makefile. There are two reasons for this:

1. When another user executes *make*, they might happen to have an environment variable of the same name (which they have set for some unrelated purpose) with a non-empty value.
2. A different implementation of *make* (or a later version of the same implementation) might have a non-empty value for the macro in its default set.

This is one aspect of a more general problem, which is that any macro that is not one for which this standard requires a default value, and is not explicitly set either in the makefile or on the *make* command line, can have an unexpected value (or unexpectedly not exist) when the makefile is used by a different user or with a different *make* implementation.

For this reason, it is recommended that makefile authors do not design makefile schemes in which values for non-standard macros are obtained from the user's environment variables. Safer methods of allowing users to configure macro values include:

- Setting the macros to default values in a *make* include file where the user can edit the values.
- Executing *make* from one or more wrapper scripts which set macro values on the command line (and which do not obtain those values from environment variables).

Makefile authors who follow this recommendation may wish to check for any macros they have overlooked by using a *make* implementation that provides, as an extension, a command-line option that causes *make* to report attempts to expand (or append to) macros that do not exist. Users of makefiles written by others can also benefit from the use of such an option to detect the opposite problem (where the author had a macro being set from an environment variable but the user does not have the variable set). This can avoid misbehaviors that would otherwise be hard to debug, such as a file-processing utility reading from standard input because it was not given any pathnames to process.

Makefile authors who choose not to follow the recommendation can minimize the risk of misbehavior by ensuring all non-standard macros have names that begin with a suitable project-specific prefix.

Use of the **−e** option is strongly discouraged, as it makes the problem discussed above even more likely by introducing the possibility of unexpected values occurring even for macros set in the makefile. If a specific macro needs a value from the environment to override a value set in the makefile, it is safer to set just that macro on the command line, using for example `make MYPROJ_FOO="$MYPROJ_FOO"`. Alternatively, the makefile can be modified to use the `?=` assignment operator for that macro.

Delayed-expansion macros are evaluated when they are used rather than when they are defined. Therefore:

```
MACRO = value1
Immed ::= $(MACRO)
DELAY = $(MACRO)
MACRO = value2

target:
    echo $(Immed) $(DELAY)
```

would produce `"value1 value2"`, since **Immed** was expanded while **MACRO** was *value1*,

but **DELAY** was not expanded until it was needed in the *echo* command line when **MACRO** was *value2*.

Because the behavior of the += assignment differs depending on whether the macro being appended to is a delayed-expansion macro or an immediate-expansion macro, it is recommended that when both macro types are used in a set of multiple makefiles and include files, a naming convention is adopted to distinguish the two macro types. This avoids any confusion about whether += will append the expanded or unexpanded value. A suitable convention might be to name delayed-expansion macros using uppercase and underscore characters and immediate-expansion macros using a name that contains at least one lowercase character.

Some historical applications have been known to intermix *target_name* and *macro=name* operands on the command line, expecting that all of the macros are processed before any of the targets are dealt with. Conforming applications do not do this, although some backwards-compatibility support may be included in some implementations.

The following characters in filenames may give trouble: `'='`, `':'`, `` '`' ``, single-quote, and `'@'`. In include filenames, pattern matching characters and `'"'` should also be avoided, as they may be treated as special by some implementations.

For inference rules, the descriptions of $< and $? seem similar. However, an example shows the minor difference. In a makefile containing:

```
foo.o: foo.h
```

if **foo.h** is newer than **foo.o**, yet **foo.c** is older than **foo.o**, the built-in rule to make **foo.o** from **foo.c** is used, with $< equal to **foo.c** and $? equal to **foo.h**. If **foo.c** is also newer than **foo.o**, $< is equal to **foo.c** and $? is equal to **foo.h foo.c**.

As a consequence of the general rules for target updating, a useful special case is that if a target has no prerequisites and no commands, and the target of the rule is a nonexistent file, then *make* acts as if this target has been updated whenever its rule is run.

Note: This implies that all targets depending on this one will always have their commands run.

Shell command sequences like `make; cp original copy; make` may have problems on filesystems where the timestamp resolution is the minimum (1 second) required by the standard and where *make* considers identical timestamps to be up-to-date. Conversely, rules like `copy: original; cp -p original copy` will result in redundant work on *make* implementations that consider identical timestamps to be out-of-date.

The requirements relating to dynamic include files (ones that have rules to create or update them) allow two different methods of implementing them, as explained in RATIONALE below. In order to write a set of makefiles and include files (both dynamic and static) that work with both methods, applications need to ensure the following:

- Rules containing commands for creating dynamic include files (and any prerequisites) precede the include line that will cause the file to be read.
- Include lines and rules for creating dynamic include files do not depend on the contents of any earlier dynamic include file. For example, defining a macro in a dynamic include file and using that macro in a later include line should be avoided (unless the later include line is itself inside the dynamic include file).
- One of the rules used in creating a dynamic include file, or a dynamic prerequisite of an include file, contains commands to create its target and does not ignore creation failure.

Note that although the first point above appears at first sight to preclude a dynamic include file

106241 that defines its own prerequisites, this can be achieved by having two include lines that name
106242 the same file (provided the rules in the file do not contain command lines), as shown in
106243 EXAMPLES.

106244 This standard does not specify precedence between macro definition and include directives.
106245 Thus, the behavior of:

106246 `include =foo.mk`

106247 is unspecified. To define a variable named include, either the white space before the <equal-
106248 sign> should be removed, or another macro should be used, as in:

106249 `INCLUDE_NAME = include`
106250 `$(INCLUDE_NAME) =foo.mk`

106251 On the other hand, if the intent is to include a file which starts with an <equal-sign>, either the
106252 filename should be changed to `./=foo.mk`, or the makefile should be written as:

106253 `INCLUDE_FILE = =foo.mk`
106254 `include $(INCLUDE_FILE)`

106255 It is important to be careful when using parallel execution (the **–j** option) and archives. If
106256 multiple `$(AR)` commands run at the same time on the same archive file, they will not know
106257 about each other and can corrupt the file. If the **–j** option is used, it is necessary to use **.WAIT** in
106258 between archive member prerequisites to prevent this (see EXAMPLES).

106259 **EXAMPLES**

106260 1. The following command:

106261 `make`

106262 makes the first target found in the makefile.

106263 2. The following command:

106264 `make junk`

106265 makes the target **junk**.

106266 3. The following makefile says that **pgm** depends on two files, **a.o** and **b.o**, and that they in
106267 turn depend on their corresponding source files (**a.c** and **b.c**), and a common file **incl.h**:

```
106268 .POSIX:
106269 pgm: a.o b.o
106270     c17 a.o b.o -o pgm
106271 a.o: incl.h a.c
106272     c17 -c a.c
106273 b.o: incl.h b.c
106274     c17 -c b.c
```

106275 4. The following is an extended version of the previous example that generates *make* include
106276 files containing the prerequisites for each object file. The include file also defines its own
106277 prerequisites, and the makefile arranges that these are used by including the file twice.
106278 With implementations of *make* that create include files during parsing, the first time *make*
106279 is run the file does not exist but the use of the <hyphen-minus> prefix on the first include
106280 line means it is ignored and is then generated by the rule preceding the second include
106281 line. On subsequent runs, if any of the source or header files previously identified has
106282 changed, the include file will be updated. There are other ways of updating the include
106283 file when headers change, but they involve recursive execution of *make*.


```
.POSIX:
.SUFFIXES: .c .d

OFILES = a.o b.o

pgm: $(OFILES)
    c17 $(OFILES) -o pgm
a.o:
    c17 -c a.c
b.o:
    c17 -c b.c

-include $(OFILES:.o=.d)
.c.d:
    +{ \
      set -o pipefail; cfile=$<; ofile=$${cfile%.c}.o; \
      printf '%s %s: %s ' "$$ofile" $@ $<; \
      c17 -E $< | LC_ALL=C sed -n \
        '/^#[[:blank:]]*[[:digit:]]/s/.*"\([^"]*\.h\)".*/\1/p' | \
        LC_ALL=C sort -u | tr '\n' ' '; \
      echo; \
    } > $@
include $(OFILES:.o=.d)
```

This example does not cope with a header file being removed (*make* will complain that it does not know how to make it), necessitating that ***.d** be removed and the *make* run again. Alternatively, this can be handled by updating the commands which generate the **.d** file so that they add an empty target rule for each of its prerequisites.

5. An example for making optimized **.o** files from **.c** files is:

```
.c.o:
    c17 -c -O 1 $*.c
```

or:

```
.c.o:
    c17 -c -O 1 $<
```

6. The most common use of the archive interface follows. Here, it is assumed that the source files are all C-language source:

```
lib.a: lib.a(file1.o) .WAIT lib.a(file2.o) .WAIT lib.a(file3.o)
    @echo lib.a is now up-to-date
```

The **.c.a** rule is used to make **file1.o**, **file2.o**, and **file3.o** and insert them into **lib.a**.

7. The treatment of escaped <newline> characters throughout the makefile is historical practice. For example, the inference rule:

```
.c.o\
:
```

works, and the macro:

```
f=  bar baz\
    biz
a:
```



```
        echo ==$f==
```

echoes "==bar baz biz==".

If $? were:

```
/usr/include/stdio.h /usr/include/unistd.h foo.h
```

then $(?D) would be:

```
/usr/include /usr/include .
```

and $(?F) would be:

```
stdio.h unistd.h foo.h
```

8. The contents of the built-in rules can be viewed by running:

```
make -p -f /dev/null 2>/dev/null
```

9. With the following makefile, `make -j 10 all` may bring `one` and `two` up-to-date in parallel despite the **.WAIT** in the prerequisite list for `foo`. This is because the **.WAIT** does not stop *make* from recursively processing `bar` and its prerequisites in parallel. However, if only `foo` is specified (`make -j 10 foo`), *make* will wait for `one` to be brought up-to-date before bringing `two` up-to-date. Note also that the **.WAIT** does not create a prerequisite relationship between `one` and `two`, so `make -j 10 two` will not build `one`.

```
all: foo bar
foo: one .WAIT two
bar: one two
foo bar one two: ; @echo $@
```

RATIONALE

The *make* utility described in this volume of POSIX.1-2024 is intended to provide the means for changing portable source code into executables that can be run on a POSIX.1-2024-conforming system. It reflects the most common features present in System V and BSD *make*s.

Historically, the *make* utility has been an especially fertile ground for vendor and research organization-specific syntax modifications and extensions. Examples include:

- Syntax supporting parallel execution (such as from various multi-processor vendors, GNU, and others)
- Additional ``operators'' separating targets and their prerequisites (System V, BSD, and others)
- Modifications of the meaning of internal macros when referencing libraries (BSD and others)
- Using a single instance of the shell for all of the command lines of the target (BSD and others)
- Allowing <space> characters as well as <tab> characters to delimit command lines (BSD)
- Adding C preprocessor-style ``include'' and ``ifdef'' constructs (System V, GNU, BSD, and others)
- Remote execution of command lines (Sprite and others)
- Specifying additional special targets (BSD, System V, and most others)

- Specifying an alternate shell to use to process commands.

Additionally, many vendors and research organizations have rethought the basic concepts of *make*, creating vastly extended, as well as completely new, syntaxes. Each of these versions of *make* fulfills the needs of a different community of users; it is unreasonable for this volume of POSIX.1-2024 to require behavior that would be incompatible (and probably inferior) to historical practice for such a community.

In similar circumstances, when the industry has enough sufficiently incompatible formats as to make them irreconcilable, this volume of POSIX.1-2024 has followed one or both of two courses of action. Commands have been renamed (*cksum*, *echo*, and *pax*) and/or command line options have been provided to select the desired behavior (*grep*, *od*, and *pax*).

Because the syntax specified for the *make* utility was, by and large, a subset of the syntaxes accepted by almost all versions of *make* when the original IEEE Std 1003.2-1992 shell and utilities standard was being developed, it was decided that it would be counter-productive to change the name. And since the makefile itself is a basic unit of portability, it would not be completely effective to reserve a new option letter, such as *make* **−P**, to achieve the portable behavior. Therefore, the special target **.POSIX** was added to the makefile, allowing users to specify ``standard'' behavior. This special target does not preclude extensions in the *make* utility, nor does it preclude such extensions being used by the makefile specifying the target; it does, however, preclude any extensions from being applied that could alter the behavior of previously valid syntax; such extensions must be controlled via command line options or new special targets. It is incumbent upon portable makefiles to specify the **.POSIX** special target in order to guarantee that they are not affected by local extensions.

The portable version of *make* described in this reference page is not intended to be the state-of-the-art software generation tool and, as such, some newer and more leading-edge features have not been included. An attempt has been made to describe the portable makefile in a manner that does not preclude such extensions as long as they do not disturb the portable behavior described here.

When the **−n** option is specified, it is always added to *MAKEFLAGS*. This allows a recursive *make* **−n** *target* to be used to see all of the actions that would be taken to update *target*.

The definition of *MAKEFLAGS* allows both the System V letter string and the BSD command line formats. The two formats are sufficiently different to allow implementations to support both without ambiguity.

Early proposals stated that an ``unquoted'' <number-sign> was treated as the start of a comment. The *make* utility does not pay any attention to quotes. A <number-sign> starts a comment regardless of its surroundings.

The text about ``other implementation-defined pathnames may also be tried'' in addition to **./makefile** and **./Makefile** is to allow such extensions as **SCCS/s.Makefile** and other variations. It was made an implementation-defined requirement (as opposed to unspecified behavior) to highlight surprising implementations that might select something unexpected like **/etc/Makefile**. XSI-conformant systems also try **./s.makefile**, **SCCS/s.makefile**, **./s.Makefile**, and **SCCS/s.Makefile**.

The default rules are based on System V. The default **CC=** value is *c17* instead of *cc* because this volume of POSIX.1-2024 does not standardize the utility named *cc*. Thus, every conforming application would be required to define **CC=***c17* to expect to run. There is no advantage conferred by the hope that the makefile might hit the ``preferred'' compiler because this cannot be guaranteed to work. Also, since the portable makescript can only use the *c17* options, no advantage is conferred in terms of what the script can do. It is a quality-of-implementation issue

as to whether *c17* is as valuable as *cc*.

Implementations are permitted to include any macro of their choosing in the default set. However, they are encouraged to keep such additions to a minimum in order to reduce the risk of name clashes with user macros.

Implementations are encouraged to provide, as an extension, a command-line option that causes *make* to report attempts to expand (or append to) macros that do not exist. See APPLICATION USAGE for the intended use cases of such an option.

The **−d** option to *make* is frequently used to produce debugging information, but is too implementation-defined to add to this volume of POSIX.1-2024.

The **−p** option is not passed in *MAKEFLAGS* on most historical implementations and to change this would cause many implementations to break without sufficiently increased portability.

Commands that begin with a <plus-sign> (`'+'`) are executed even if the **−n** option is present. Based on the GNU version of *make*, the behavior of **−n** when the <plus-sign> prefix is encountered has been extended to apply to **−q** and **−t** as well. The System V convention of forcing command execution with **−n** when the command line of a target expands the *MAKE* macro was not adopted in earlier versions of this standard, but it is now required because it has become widespread existing practice.

The double <colon> in the target rule format is supported in BSD systems to allow more than one target line containing the same target name to have commands associated with it. Since this is not functionality described in the SVID or XPG3 it has been allowed as an extension, but not mandated.

The default rules are provided with text specifying that the built-in rules shall be the same as if the listed set were used. The intent is that implementations should be able to use the rules without change, but will be allowed to alter them in ways that do not affect the primary behavior.

One point of discussion was whether to drop the default rules list from this volume of POSIX.1-2024. They provide convenience, but do not enhance portability of applications. The prime benefit is in portability of users who wish to type *make command* and have the command build from a **command.c** file.

The historical *MAKESHELL* feature, and related features provided by other *make* implementations, were omitted. In some implementations it is used to let a user override the shell to be used to run *make* commands. This was confusing; for a portable *make*, the shell should be chosen by the makefile writer. Further, a makefile writer cannot require an alternate shell to be used and still consider the makefile portable. While it would be possible to standardize a mechanism for specifying an alternate shell, existing implementations do not agree on such a mechanism, and makefile writers can already invoke an alternate shell by specifying the shell name in the rule for a target; for example:

```
python −c "foo"
```

The *make* utilities in most historical implementations process the prerequisites of a target in left-to-right order, and the makefile format requires this. It supports the standard idiom used in many makefiles that produce *yacc* programs; for example:

```
foo: y.tab.o lex.o main.o
    $(CC) $(CFLAGS) −o $@ y.tab.o lex.o main.o
```

In this example, if *make* chose any arbitrary order, the **lex.o** might not be made with the correct **y.tab.h**. Although there may be better ways to express this relationship, it is widely used

106458 historically. Implementations that desire to update prerequisites in parallel should require an
106459 explicit extension to *make* or the makefile format to accomplish it, as described previously.

106460 The algorithm for determining a new entry for target rules is partially unspecified. Some
106461 historical *make*s allow comment lines (including blank and empty lines) within the collection of
106462 commands marked by leading <tab> characters. A conforming makefile must ensure that each
106463 command starts with a <tab>, but implementations are free to ignore comments without
106464 triggering the start of a new entry.

106465 The ASYNCHRONOUS EVENTS section includes having SIGTERM and SIGHUP, along with
106466 the more traditional SIGINT and SIGQUIT, remove the current target unless directed not to do
106467 so. SIGTERM and SIGHUP were added to parallel other utilities that have historically cleaned
106468 up their work as a result of these signals. When *make* receives any signal other than SIGQUIT, it
106469 is required to resend itself the signal it received so that it exits with a status that reflects the
106470 signal. The results from SIGQUIT are partially unspecified because, on systems that create a file
106471 named **core** upon receipt of SIGQUIT, the **core** file from *make* would conflict with a **core** file from
106472 the command that was running when the SIGQUIT arrived. The main concern was to prevent
106473 damaged files from appearing up-to-date when *make* is rerun.

106474 The **.PRECIOUS** special target was extended to affect all targets globally (by specifying no
106475 prerequisites). The **.IGNORE** and **.SILENT** special targets were extended to allow prerequisites;
106476 it was judged to be more useful in some cases to be able to turn off errors or echoing for a list of
106477 targets than for the entire makefile. These extensions to *make* in System V were made to match
106478 historical practice from the BSD *make*.

106479 Macros are not exported to the environment of commands to be run. This was never the case in
106480 any historical *make* and would have serious consequences. The environment is the same as the
106481 environment to *make* except that *MAKEFLAGS* and macros defined on the *make* command line
106482 are added, and except that macros defined by the *MAKEFLAGS* environment variable and
106483 macros defined in the makefile(s) may update the value of an existing environment variable
106484 (other than *SHELL*).

106485 Some implementations do not use *system*() for all command lines, as required by the portable
106486 makefile format; as a performance enhancement, they select lines without shell metacharacters
106487 for direct execution by *execve*(). There is no requirement that *system*() be used specifically, but
106488 merely that the same results be achieved. The metacharacters typically used to bypass the direct
106489 *execve*() execution have been any of:

106490 `= | ^ ( ) ; & < > * ? [ ] : $ ` ' " \ \n`

106491 The default in some advanced versions of *make* is to group all the command lines for a target and
106492 execute them using a single shell invocation; the System V method is to pass each line
106493 individually to a separate shell. The single-shell method has the advantages in performance and
106494 the lack of a requirement for many continued lines. However, converting to this newer method
106495 has caused portability problems with many historical makefiles, so the behavior with the POSIX
106496 makefile is specified to be the same as that of System V. It is suggested that the special target
106497 **.ONESHELL** be used as an implementation extension to achieve the single-shell grouping for a
106498 target or group of targets.

106499 Novice users of *make* have had difficulty with the historical need to start commands with a
106500 <tab>. Since it is often difficult to discern differences between <tab> and <space> characters on
106501 terminals or printed listings, confusing bugs can arise. In early proposals, an attempt was made
106502 to correct this problem by allowing leading <blank> characters instead of <tab> characters.
106503 However, implementors reported many makefiles that failed in subtle ways following this
106504 change, and it is difficult to implement a *make* that unambiguously can differentiate between
106505 macro and command lines. There is extensive historical practice of allowing leading <space>

characters before macro definitions. Forcing macro lines into column 1 would be a significant backwards-compatibility problem for some makefiles. Therefore, historical practice was restored.

There is substantial variation in the handling of include lines by different implementations. However, there is enough commonality for the standard to be able to specify a minimum set of requirements that allow the feature to be used portably. Known variations have been explicitly called out as unspecified behavior in the description.

The System V dynamic dependency feature was not included. It would support:

```
cat: $$@.c
```

that would expand to;

```
cat: cat.c
```

This feature exists only in the new version of System V *make* and, while useful, is not in wide usage. This means that macros are expanded twice for prerequisites: once at makefile parse time and once at target update time.

Consideration was given to adding metarules to the POSIX *make*. This would make **%.o: %.c** the same as **.c.o:**. This is quite useful and available from some vendors, but it would cause too many changes to this *make* to support. It would have introduced rule chaining and new substitution rules. However, the rules for target names have been set to reserve the `'%'` and `'"'` characters. These are traditionally used to implement metarules and quoting of target names, respectively. Implementors are strongly encouraged to use these characters only for these purposes.

A request was made to extend the suffix delimiter character from a <period> to any character. The metarules feature in newer *make*s solves this problem in a more general way. This volume of POSIX.1-2024 is staying with the more conservative historical definition.

The standard output format for the −**p** option is not described because it is primarily a debugging option and because the format is not generally useful to programs. In historical implementations the output is not suitable for use in generating makefiles. The −**p** format has been variable across historical implementations. Therefore, the definition of −**p** was only to provide a consistently named option for obtaining *make* script debugging information.

Some historical implementations have not cleared the suffix list with −**r**.

Implementations should be aware that some historical applications have intermixed *target_name* and *macro=value* operands on the command line, expecting that all of the macros are processed before any of the targets are dealt with. Conforming applications do not do this, but some backwards-compatibility support may be warranted.

Empty inference rules are specified with a <semicolon> command rather than omitting all commands, as described in an early proposal. The latter case has no traditional meaning and is reserved for implementation extensions, such as in GNU *make*.

Earlier versions of this standard defined comment lines only as lines with `'#'` as the first character. Many places then talked about comments, blank lines, and empty lines; but some places inadvertently only mentioned comments when blank lines and empty lines had also been accepted in all known implementations. The standard now defines comment lines to be blank lines, empty lines, and lines starting with a `'#'` character and explicitly lists cases where blank lines and empty lines are not acceptable.

On most historic systems, the *make* utility considered a target with a prerequisite that had an identical timestamp as up-to-date. One implementation of *make* treated it as out-of-date. Note that up-to-date and out-of-date are antonyms. The standard now allows either behavior, but

implementations are encouraged to treat such targets as out-of-date. This is especially important on file systems where the timestamp resolution is the minimum (1 second) required by the standard. All implementations of *make* should make full use of the finest timestamp resolution available on the file systems holding targets and prerequisites to ensure that targets are up-to-date even for prerequisite files with timestamps that were updated within the same second. However, if the timestamp resolutions of the file systems containing a target and a prerequisite are different, the timestamp with the more precise resolution should be rounded down to the resolution of the less precise timestamp for the comparison.

The traditional semantics of delayed-expansion macros have often been the source of subtle bugs for makefile writers not aware of those semantics. Furthermore, in implementations that support an extension of assigning the output of an arbitrary command to a macro definition, the use of delayed-expansion macros could result in an undesirable growth in execution time, as each use of the macro would re-run the arbitrary command. Historically, several implementations independently developed a form of immediate expansion, usually via the operator `":="`, so that execution of an arbitrary command happens once at the definition of the macro rather than each use of the macro; however, there are subtle differences in the expansion rules of those various implementations when the expanded value of *string2* contained a `'$'`. Other implementations used the operator `":="` for conditional expansion, altogether unrelated to immediate-expansion macro definition.

The standard developers felt that immediate-expansion semantics were useful enough to standardize, but requiring the semantics of any one implementation of `":="` would cause confusion in makefiles written for other implementation semantics, necessitating a reader to determine if `.POSIX:` had been specified at the beginning of the file (or worse, at the beginning of some other file that then includes the fragment in question) to know which semantics would be in use. Therefore, the standard developers opted to require two new operators, `":::="` and `"::::="`, with specific semantics; the `":::="` operator has semantics closest to the GNU *make* implementation of `":="`, where `'$'` characters occurring in the immediate expansion of *string2* are not further expanded in subsequent use of the macro, and the `"::::="` operator has semantics closest to the BSD *make* and *smake* implementations of `":="`, where immediate expansion is performed when assigning to a delayed-expansion macro and `"$$"` is preserved. It was felt that other implementations could easily support the required semantics.

Implementations that previously provided `":="` as an extension are encouraged to leave this extension intact, with no change in the implementation's particular semantics, to avoid breaking non-portable makefiles that had been targeting that particular implementation. A portable makefile, with `.POSIX:` specified at the beginning, should not use the `":="` operator.

Traditionally, constructs such as

```
DIR: FORCE
        (commands)
FORCE:
```

were used to allow `make DIR` to always run `(commands)`; however, this depended on the user never creating a file named **FORCE**. The addition of the **.PHONY** special target provides a more efficient manner of providing a target whose commands are always run, and where the user cannot create a file that influences the behavior in an unexpected manner.

This standard allows two different methods of creating include files or bringing them up-to-date, reflecting established practice in SunPro *make* and GNU *make*. The former performs this action during parsing, before the include file is opened. The latter delays performing the action until after all makefiles have been read. Implementors who opt for the ``delayed remaking'' method should be aware of the following potential issues:

- Diagnostic messages about missing include files must be deferred until the final exit status is known. Note that this is a conformance issue, not just a quality of implementation issue.
- If the way *make* handles using updated include file contents is to start over after include files have been made up-to-date, it is possible for a poorly written makefile to cause *make* to enter a sequence of restarts where nothing changes each time, resulting in the sequence continuing indefinitely unless the situation is detected. Implementors are encouraged to include a mechanism for detecting and reporting this, rather than allowing *make* to consume an arbitrary amount of system resource until it is forcibly terminated.
- If *make* uses this start-over method, makefile contents read from a pipe on standard input or from a FIFO must be copied to a temporary file, and when *make* starts over it must use this file instead.
- If *make* starts over by executing itself using the *exec* family of functions, the need to replace `'-'` or the pathnames of FIFOs with the pathnames of temporary files can lead to the *exec* call failing with an [E2BIG] error if the original execution was close to the {ARG_MAX} limit. Although this is a quality of implementation issue, not a conformance issue (since the general rules for utility errors allow utilities to fail when they encounter a variety of internal errors - see Section 1.4, on page 2462), implementors are encouraged to explore ways to prevent it, such as passing information via a temporary file instead of on the command line when an [E2BIG] error has occurred. Another solution might be to jump (e.g. using *siglongjmp*( )) back to the start of *main*( ) as the way to start over. Making a recursive call to *main*( ) is not recommended, as that would run into the stack limit if sufficiently many restarts are needed.

This standard specifies that a non-existent include file is first created if possible, and only if not possible can other directories be searched. Historical versions of GNU *make* first searched the include directories, then attempted to create the include file. This behavior was not considered suitable for standardization as it means writers of portable applications have to use absolute pathnames for all include files that need to be created via a rule (because they can never be sure what relative pathnames are safe to use, since a file with the same relative pathname might happen to exist in one of the searched directories when installing the application on a new system). Note, however, that this only applies to directories searched by default. If an application uses an extension to specify that one or more directories are searched, this standard does not place any constraints on when the specified directories are searched.

This standard specifies a way for portable applications to request parallel updating of targets with commands by using the **–j** *maxjobs* option. This feature is described in terms of a token pool initially containing up to *maxjobs* – 1 tokens. Note that this is not intended to prescribe a particular implementation design; the usual ``as if'' rule applies.

Implementations are permitted to silently limit the pool size for a few reasons, including:

- Implementations that do not support parallelism can support the **–j** option by simply ignoring the option (other than passing it to sub-*make* invocations via the *MAKEFLAGS* environment variable). In effect, such an implementation silently restricts the size of the token pool to zero (and therefore need not create a token pool).
- Some historical implementations dynamically limit the token pool size based on the current system load to avoid overloading the system.
- Implementations may want to limit the token pool size based on the number of processors available.

- Implementations may want to limit the token pool size based on resource limits.

Limiting the pool size does not change the value of *maxjobs* that is passed to sub-*make* invocations via the *MAKEFLAGS* environment variable.

When a different *maxjobs* value is passed to a sub-*make*, some historical *make* implementations created a separate pool of tokens while other historical *make* implementations continued to obtain tokens from the invoking *make* but limited the number of tokens held at a time to the new value of *maxjobs* − 1. Both behaviors are believed to have merit in different situations: the former gives a sub-*make* complete control the amount of parallelism, while the latter allows the user to control the overall system load. This standard permits either behavior.

This standard calls for a token pool of size *maxjobs* − 1, and for removal from that pool only for the second and subsequent tasks in a set of parallel tasks. This design was chosen because this is effectively what existing implementations do, and also because the token consumed by a parallel task that invokes a sub-*make* is effectively lent to the sub-*make*. Lending the token to the sub-*make* has the following advantages:

- It prevents the sub-*make* from being completely idle due to token starvation, allowing it to always make some progress regardless of how many tokens other sub-*make* invocations have consumed.
- It prevents token pool exhaustion caused by a long chain of sub-*make* invocations. If the token consumed by the invoking rule was not effectively lent to the sub-*make*, then the pool would be exhausted by a chain of sub-*make* invocations that is *maxjobs* long. Such a chain would never accomplish any work, and would thus never complete.

When a rule invokes multiple sub-*make* processes asynchronously (for example by using an asynchronous list in the shell), some implementations allow each sub-*make* to execute at least one rule even though this would cause the total number of parallel rule executions across all *make* instances to exceed *maxjobs* (after discounting the rules that execute sub-*make* processes). This behavior may not be ideal, but it is easier to implement and is unlikely to cause problems in practice because applications typically do not have any rules that invoke multiple sub-*make* processes asynchronously. For this reason the behavior is unspecified if a rule executes multiple sub-*make* processes asynchronously.

When multiple sub-*make* processes are running in parallel there is no requirement placed on the ordering of output from these processes. Some implementations of *make* attempt to serialize output from each sub-*make*; others make no such attempt. If diagnostic messages from failed commands are intermixed, the usual way to deal with this is to repeat the *make* without **−j** (or with **−j** 1) so that intermixing will not occur.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a **<**newline**>** character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

Some implementations of *make* include an *export* directive to add specified *make* variables to the environment. This may be considered for standardization in a future version.

A future version of this standard may add a command-line option that causes *make* to report attempts to expand (or append to) macros that do not exist.

A future version of this standard may require that a target with a prerequisite with an identical timestamp is considered out-of-date.

106688 **SEE ALSO**

106689 Chapter 2 (on page 2472), *ar*, *c17*, *get*, *lex*, *sccs*, *sh*, *yacc*

106690 XBD Section 6.1 (on page 117), Chapter 8 (on page 167), Section 12.2 (on page 215)

106691 XSH *exec*, *system*( )

106692 **CHANGE HISTORY**

106693 First released in Issue 2.

106694 **Issue 5**

106695 The FUTURE DIRECTIONS section is added.

106696 **Issue 6**

106697 This utility is marked as part of the Software Development Utilities option.

106698 The Open Group Corrigendum U029/1 is applied, correcting a typographical error in the
106699 SPECIAL TARGETS section.

106700 In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from
106701 ``otherwise, the home directory of a user of that name is examined'' to ``otherwise, the value of
106702 *PROJECTDIR* is treated as a user name and that user's initial working directory is examined''.

106703 It is specified whether the command line is related to the makefile or to the *make* command, and
106704 the macro processing rules are updated to align with the IEEE P1003.2b draft standard.

106705 The normative text is reworded to avoid use of the term ``must'' for application requirements.

106706 PASC Interpretation 1003.2 #193 is applied.

106707 **Issue 7**

106708 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not
106709 apply.

106710 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

106711 Include lines in makefiles are introduced.

106712 Austin Group Interpretation 1003.1-2001 #131 is applied, changing the **Makefile Execution**
106713 section.

106714 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0121 [257] is applied.

106715 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0122 [509], XCU/TC2-2008/0123
106716 [584], XCU/TC2-2008/0124 [857], XCU/TC2-2008/0125 [505], XCU/TC2-2008/0126 [584],
106717 XCU/TC2-2008/0127 [505], XCU/TC2-2008/0128 [865], XCU/TC2-2008/0129 [693],
106718 XCU/TC2-2008/0130 [602], XCU/TC2-2008/0131 [848], XCU/TC2-2008/0132 [763],
106719 XCU/TC2-2008/0133 [857], XCU/TC2-2008/0134 [866], XCU/TC2-2008/0135 [525],
106720 XCU/TC2-2008/0136 [848], XCU/TC2-2008/0137 [769], XCU/TC2-2008/0138 [525],
106721 XCU/TC2-2008/0139 [769], XCU/TC2-2008/0140 [505], XCU/TC2-2008/0141 [693],
106722 XCU/TC2-2008/0142 [505], XCU/TC2-2008/0143 [857], and XCU/TC2-2008/0144 [693,865] are
106723 applied.

106724 **Issue 8**

106725 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of
106726 filenames containing any bytes that have the encoded value of a <newline> character.

106727 Austin Group Defects 330, 1417, 1422, 1709, and 1710 are applied, adding new forms of macro
106728 assignment using the `":::="`, `"?="`, and `"+="` operators.

106729 Austin Group Defect 333 is applied, adding support for ``silent includes'' using **–include**.

Austin Group Defects 336 and 1711 are applied, specifying the behavior when *string1* in a macro expansion contains a macro expansion.

Austin Group Defect 337 is applied, adding a new form of macro assignment using the `"!="` operator.

Austin Group Defects 373 and 1417 are applied, changing the set of characters that portable applications can use in macro names to the entire portable filename character set (thus adding <hyphen-minus> to the set that could previously be used).

Austin Group Defects 514 and 1520 are applied, adding the $+ and $^ internal macros.

Austin Group Defect 518 is applied, allowing multiple files to be specified on an **include** line.

Austin Group Defects 519, 1712, and 1715 are applied, adding support for pattern macro expansions.

Austin Group Defects 523, 1708, and 1749 are applied, adding the **.PHONY** special target.

Austin Group Defect 875 is applied, clarifying the requirements for inference rules.

Austin Group Defect 1104 is applied, changing ``***s2.a***'' to ``***.s2.a***''.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1141 is applied, changing ``core files'' to ``a file named core''.

Austin Group Defect 1155 is applied, clarifying the handling of the *MAKE* macro.

Austin Group Defect 1325 is applied, adding requirements relating to the creation of include files.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1419 is applied, updating the **.SCCS_GET** default rule.

Austin Group Defect 1420 is applied, clarifying where internal macros can be used.

Austin Group Defect 1421 is applied, changing the APPLICATION USAGE section.

Austin Group Defects 1424, 1658, 1690, 1701, 1702, 1703, 1704, 1707, 1719, 1720, 1721, 1722, and 1750 are applied, making various minor editorial wording changes.

Austin Group Defects 1436, 1437, 1652, 1660, 1661, and 1733 are applied, adding the **–j** *maxjobs* option and the **.NOTPARALLEL** and **.WAIT** special targets, and changing the **–n** option.

Austin Group Defects 1471 and 1513 are applied, adding a new form of macro assignment using the `":::="` operator.

Austin Group Defect 1479 is applied, clarifying the requirements for default rules and macro values.

Austin Group Defect 1492 is applied, changing the EXIT STATUS section.

Austin Group Defect 1505 is applied, clarifying the requirements for expansion of macros that do not exist.

Austin Group Defect 1510 is applied, correcting a typographic error in the RATIONALE section.

Austin Group Defect 1549 is applied, clarifying the requirements for an escaped <newline> in a command line.

Austin Group Defect 1615 is applied, allowing target names to contain slashes and hyphens.

Austin Group Defect 1626 is applied, adding the *CURDIR* macro.

Austin Group Defect 1631 is applied, adding information about use of the **–j** option with the **.c.a** default rule to the APPLICATION USAGE and EXAMPLES sections.

Austin Group Defect 1650 is applied, changing the few occurrences of ``dependencies'' to use the more common ``prerequisites''.

Austin Group Defect 1653 is applied, clarifying the difference between how *MAKEFLAGS* is parsed compared to shell commands that use the *make* utility.

Austin Group Defects 1654 and 1655 are applied, changing the APPLICATION USAGE section.

Austin Group Defect 1656 is applied, changing the NAME section.

Austin Group Defect 1657 is applied, moving some requirements unrelated to makefile syntax from the Makefile Syntax subsection to the beginning of the EXTENDED DESCRIPTION section.

Austin Group Defect 1689 is applied, removing some redundant wording from the DESCRIPTION section.

Austin Group Defect 1692 is applied, allowing *make*, when invoked with the **–q** or **–t** option, to execute command lines (without a **<**plus-sign**>** prefix) that expand the *MAKE* macro.

Austin Group Defect 1693 is applied, changing ``command lines'' to ``execution lines'' in the description of the **–s** option.

Austin Group Defect 1694 is applied, changing ``in the order they appear'' to ``in the order specified'' in the OPERANDS section.

Austin Group Defect 1696 is applied, changing the STDOUT section.

Austin Group Defect 1697 is applied, changing the RATIONALE and FUTURE DIRECTIONS sections.

Austin Group Defect 1698 is applied, changing ``of a target'' to ``of the target'' in the EXTENDED DESCRIPTION section.

Austin Group Defect 1699 is applied, addressing some inconsistencies in the use of the term ``rules''.

Austin Group Defect 1706 is applied, removing a line from the format specified for target rules.

Austin Group Defect 1714 is applied, changing ``beginning of the line'' to ``beginning of the value''.

Austin Group Defect 1716 is applied, changing the typographic convention used for variable elements within target names, in particular the inference rule suffixes ***s1*** and ***s2***.

Austin Group Defect 1723 is applied, adding historical context to a paragraph in the RATIONALE section.

Austin Group Defect 1772 is applied, clarifying the ASYNCHRONOUS EVENTS section.

NAME

man — display system documentation

SYNOPSIS

UP

```
man [-k] name...
```

DESCRIPTION

The *man* utility shall write information about each of the *name* operands. If *name* is the name of a standard utility, *man* at a minimum shall write a message describing the syntax used by the standard utility, its options, operands, environment variables affecting its execution, and its list of exit status codes. If more information is available, the *man* utility shall provide it in an implementation-defined manner.

An implementation may provide information for values of *name* other than the standard utilities. Standard utilities that are listed as optional and that are not supported by the implementation either shall cause a brief message indicating that fact to be displayed or shall cause a full display of information as described previously.

OPTIONS

The *man* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

–k Interpret *name* operands as keywords to be used in searching a utilities summary database that contains a brief purpose entry for each standard utility and write lines from the summary database that match any of the keywords. The keyword search shall produce results that are the equivalent of the output of the following command:

```
grep -Ei '
name
name
...
' summary-database
```

This assumes that the *summary-database* is a text file with a single entry per line; this organization is not required and the example using *grep* **–Ei** is merely illustrative of the type of search intended. The purpose entry to be included in the database shall consist of a terse description of the purpose of the utility.

OPERANDS

The following operand shall be supported:

name A keyword or the name of a standard utility. When **–k** is not specified and *name* does not represent one of the standard utilities, the results are unspecified.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *man*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and in the summary database). The value of *LC_CTYPE* need not affect the format of the information written about the *name* operands.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

PAGER Determine an output filtering command for writing the output to a terminal. Any string acceptable as a *command_string* operand to the *sh* **−c** command shall be valid. When standard output is a terminal device, the reference page output shall be piped through the command. If the *PAGER* variable is null or not set, the command shall be either *more* or another paginator utility documented in the system documentation.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The *man* utility shall write text describing the syntax of the utility *name*, its options and its operands, or, when **−k** is specified, lines from the summary database. The format of this text is implementation-defined.

STDERR

The standard error shall be used for diagnostic messages, and may also be used for informational messages of unspecified format.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

It is recognized that the *man* utility is only of minimal usefulness as specified. The opinion of the standard developers was strongly divided as to how much or how little information *man* should be required to provide. They considered, however, that the provision of some portable way of accessing documentation would aid user portability. The arguments against a fuller specification were:

- Large quantities of documentation should not be required on a system that does not have excess disk space.
- The current manual system does not present information in a manner that greatly aids user portability.
- A ``better help system'' is currently an area in which vendors feel that they can add value to their POSIX implementations.

The **−f** option was considered, but due to implementation differences, it was not included in this volume of POSIX.1-2024.

The description was changed to be more specific about what has to be displayed for a utility. The standard developers considered it insufficient to allow a display of only the synopsis without giving a short description of what each option and operand does.

The ``purpose'' entry to be included in the database can be similar to the section title (less the numeric prefix) from this volume of POSIX.1-2024 for each utility. These titles are similar to those used in historical systems for this purpose.

See *mailx* for rationale concerning the default paginator.

The caveat in the *LC_CTYPE* description was added because it is not a requirement that an implementation provide reference pages for all of its supported locales on each system; changing *LC_CTYPE* does not necessarily translate the reference page into another language. This is equivalent to the current state of *LC_MESSAGES* in POSIX.1-2024—locale-specific messages are not yet a requirement.

The historical *MANPATH* variable is not included in POSIX because no attempt is made to specify naming conventions for reference page files, nor even to mandate that they are files at all. On some implementations they could be a true database, a hypertext file, or even fixed strings within the *man* executable. The standard developers considered the portability of reference pages to be outside their scope of work. However, users should be aware that *MANPATH* is implemented on a number of historical systems and that it can be used to tailor the search pattern for reference pages from the various categories (utilities, functions, file formats, and so on) when the system administrator reveals the location and conventions for reference pages on the system.

The keyword search can rely on at least the text of the section titles from these utility descriptions, and the implementation may add more keywords. The term ``section titles'' refers to the strings such as:

```
man — Display system documentation
ps — Report process status
```


FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

more

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #108 is applied, clarifying that informational messages may appear on standard error.

Issue 8

Austin Group Defect 190 is applied, marking the *man* utility as part of the User Portability Utilities option, and adding a requirement for the message it writes for a standard utility to include the environment variables affecting its execution and its list of exit status codes.

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

mesg — permit or deny messages

SYNOPSIS

```
mesg [y|n]
```

DESCRIPTION

The *mesg* utility shall control whether other users are allowed to send messages via *write*, *talk*, or other utilities to a terminal device. The terminal device affected shall be determined by searching for the first terminal in the sequence of devices associated with standard input, standard output, and standard error, respectively. With no arguments, *mesg* shall report the current state without changing it. Processes with appropriate privileges may be able to send messages to the terminal independent of the current state.

OPTIONS

None.

OPERANDS

The following operands shall be supported in the POSIX locale:

y Grant permission to other users to send messages to the terminal device.

n Deny permission to other users to send messages to the terminal device.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *mesg*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written (by *mesg*) to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If no operand is specified, *mesg* shall display the current terminal state in an unspecified format.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Receiving messages is allowed.

1 Receiving messages is not allowed.

>1 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The mechanism by which the message status of the terminal is changed is unspecified. Therefore, unspecified actions may cause the status of the terminal to change after *mesg* has successfully completed. These actions may include, but are not limited to: another invocation of the *mesg* utility, login procedures; invocation of the *stty* utility, invocation of the *chmod* utility or *chmod*( ) function, and so on.

EXAMPLES

None.

RATIONALE

The terminal changed by *mesg* is that associated with the standard input, output, or error, rather than the controlling terminal for the session. This is because users logged in more than once should be able to change any of their login terminals without having to stop the job running in those sessions. This is not a security problem involving the terminals of other users because appropriate privileges would be required to affect the terminal of another user.

The method of checking each of the first three file descriptors in sequence until a terminal is found was adopted from System V.

The file **/dev/tty** is not specified for the terminal device because it was thought to be too restrictive. Typical environment changes for the *n* operand are that write permissions are removed for *others* and *group* from the appropriate device. It was decided to leave the actual description of what is done as unspecified because of potential differences between implementations.

The format for standard output is unspecified because of differences between historical implementations. This output is generally not useful to shell scripts (they can use the exit status), so exact parsing of the output is unnecessary.

FUTURE DIRECTIONS

None.

SEE ALSO

talk, *write*

XBD Chapter 8 (on page 167)

107030 **CHANGE HISTORY**

107031 First released in Issue 2.

107032 **Issue 6**

107033 This utility is marked as part of the User Portability Utilities option.

107034 **Issue 7**

107035 The *mesg* utility is moved from the User Portability Utilities option to the Base. User Portability
107036 Utilities is now an option for interactive utilities.

107037 **Issue 8**

107038 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

mkdir — make directories

SYNOPSIS

```
mkdir [-p] [-m mode] dir...
```

DESCRIPTION

The *mkdir* utility shall create the directories specified by the operands, in the order specified.

For each *dir* operand, the *mkdir* utility shall perform actions equivalent to the *mkdir*( ) function defined in the System Interfaces volume of POSIX.1-2024, called with the following arguments:

1. The *dir* operand is used as the *path* argument.
2. The value of the bitwise-inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO is used as the *mode* argument. (If the **–m** option is specified, the value of the *mkdir*( ) *mode* argument is unspecified, but the directory shall at no time have permissions less restrictive than the **–m** *mode* option-argument.)

OPTIONS

The *mkdir* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–m *mode* Set the file permission bits of the newly-created directory to the specified *mode* value. The *mode* option-argument shall be the same as the *mode* operand defined for the *chmod* utility. In the *symbolic_mode* strings, the *op* characters `'+'` and `'-'` shall be interpreted relative to an assumed initial mode of *a=rwx*; `'+'` shall add permissions to the default mode, `'-'` shall delete permissions from the default mode.

–p Create any missing intermediate pathname components.

For each *dir* operand that does not name an existing directory, before performing the actions described in the DESCRIPTION above, the *mkdir* utility shall create any pathname components of the path prefix of *dir* that do not name an existing directory by performing actions equivalent to first calling the *mkdir*( ) function with the following arguments:

1. A pathname naming the missing pathname component, ending with a trailing <slash> character, as the *path* argument
2. The value zero as the *mode* argument

and then calling the *chmod*( ) function with the following arguments:

1. The same *path* argument as in the *mkdir*( ) call
2. The value `(S_IWUSR|S_IXUSR|~`*filemask*`)&0777` as the *mode* argument, where *filemask* is the file mode creation mask of the process (see XSH *umask*( ))

Each *dir* operand that names an existing directory shall be ignored without error.

OPERANDS

The following operand shall be supported:

dir A pathname of a directory to be created.

107079 **STDIN**

107080 Not used.

107081 **INPUT FILES**

107082 None.

107083 **ENVIRONMENT VARIABLES**

107084 The following environment variables shall affect the execution of *mkdir*:

107085 *LANG* Provide a default value for the internationalization variables that are unset or null.
107086 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
107087 variables used to determine the values of locale categories.)

107088 *LC_ALL* If set to a non-empty string value, override the values of all the other
107089 internationalization variables.

107090 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
107091 characters (for example, single-byte as opposed to multi-byte characters in
107092 arguments).

107093 *LC_MESSAGES*

107094 Determine the locale that should be used to affect the format and contents of
107095 diagnostic messages written to standard error.

107096 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

107097 **ASYNCHRONOUS EVENTS**

107098 Default.

107099 **STDOUT**

107100 Not used.

107101 **STDERR**

107102 The standard error shall be used only for diagnostic messages.

107103 **OUTPUT FILES**

107104 None.

107105 **EXTENDED DESCRIPTION**

107106 None.

107107 **EXIT STATUS**

107108 The following exit values shall be returned:

107109 0 All the specified directories were created successfully, or the **−p** option was specified and all
107110 the specified directories either already existed or were created successfully.

107111 >0 An error occurred.

107112 **CONSEQUENCES OF ERRORS**

107113 Default.

APPLICATION USAGE

The default file mode for directories is *a=rwx* (777 on most systems) with selected permissions removed in accordance with the file mode creation mask. For intermediate pathname components created by *mkdir*, the mode is the default modified by *u+wx* so that the subdirectories can always be created regardless of the file mode creation mask; if different ultimate permissions are desired for the intermediate directories, they can be changed afterwards with *chmod*.

Note that some of the requested directories may have been created even if an error occurs.

EXAMPLES

None.

RATIONALE

The System V **−m** option was included to control the file mode.

The System V **−p** option was included to create any needed intermediate directories and to complement the functionality provided by *rmdir* for removing directories in the path prefix as they become empty. Because no error is produced if any path component already exists, the **−p** option is also useful to ensure that a particular directory exists.

The functionality of *mkdir* is described substantially through a reference to the *mkdir*( ) function in the System Interfaces volume of POSIX.1-2024. For example, by default, the mode of the directory is affected by the file mode creation mask in accordance with the specified behavior of the *mkdir*( ) function. In this way, there is less duplication of effort required for describing details of the directory creation.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

chmod, *rm*, *rmdir*, *umask*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *mkdir*( ), *umask*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 7

SD5-XCU-ERN-56 is applied, aligning the **−m** option with the IEEE P1003.2b draft standard to clarify an ambiguity.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0122 [161] is applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0145 [843] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

107156 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

mkfifo — make FIFO special files

SYNOPSIS

```
mkfifo [-m mode] file...
```

DESCRIPTION

The *mkfifo* utility shall create the FIFO special files specified by the operands, in the order specified.

For each *file* operand, the *mkfifo* utility shall perform actions equivalent to the *mkfifo*( ) function defined in the System Interfaces volume of POSIX.1-2024, called with the following arguments:

1. The *file* operand is used as the *path* argument.
2. The value of the bitwise-inclusive OR of S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH is used as the *mode* argument. (If the **−m** option is specified, the value of the *mkfifo*( ) *mode* argument is unspecified, but the FIFO shall at no time have permissions less restrictive than the **−m** *mode* option-argument.)

OPTIONS

The *mkfifo* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

−m *mode* Set the file permission bits of the newly-created FIFO to the specified *mode* value. The *mode* option-argument shall be the same as the *mode* operand defined for the *chmod* utility. In the *symbolic_mode* strings, the *op* characters `'+'` and `'−'` shall be interpreted relative to an assumed initial mode of *a=rw*.

OPERANDS

The following operand shall be supported:

file A pathname of the FIFO special file to be created.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *mkfifo*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

107198 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

107199 **ASYNCHRONOUS EVENTS**
107200 Default.

107201 **STDOUT**
107202 Not used.

107203 **STDERR**
107204 The standard error shall be used only for diagnostic messages.

107205 **OUTPUT FILES**
107206 None.

107207 **EXTENDED DESCRIPTION**
107208 None.

107209 **EXIT STATUS**
107210 The following exit values shall be returned:

107211 0 All the specified FIFO special files were created successfully.

107212 >0 An error occurred.

107213 **CONSEQUENCES OF ERRORS**
107214 Default.

107215 **APPLICATION USAGE**
107216 None.

107217 **EXAMPLES**
107218 None.

107219 **RATIONALE**
107220 This utility was added to permit shell applications to create FIFO special files.

107221 The **−m** option was added to control the file mode, for consistency with the similar functionality
107222 provided by the *mkdir* utility.

107223 Early proposals included a **−p** option similar to the *mkdir* **−p** option that created intermediate
107224 directories leading up to the FIFO specified by the final component. This was removed because
107225 it is not commonly needed and is not common practice with similar utilities.

107226 The functionality of *mkfifo* is described substantially through a reference to the *mkfifo*( ) function
107227 in the System Interfaces volume of POSIX.1-2024. For example, by default, the mode of the FIFO
107228 file is affected by the file mode creation mask in accordance with the specified behavior of the
107229 *mkfifo*( ) function. In this way, there is less duplication of effort required for describing details of
107230 the file creation.

107231 **FUTURE DIRECTIONS**
107232 If this utility is directed to create a new directory entry that contains any bytes that have the
107233 encoded value of a <newline> character, implementations are encouraged to treat this as an
107234 error. A future version of this standard may require implementations to treat this as an error.

107235 **SEE ALSO**
107236 *chmod*, *umask*

107237 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

107238 XSH *mkfifo*( )

CHANGE HISTORY

First released in Issue 3.

Issue 6

The **–m** option is aligned with the IEEE P1003.2b draft standard to clarify an ambiguity.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

more — display files on a page-by-page basis

SYNOPSIS

UP

```
more [-ceisu] [-n number] [-p command] [-t tagstring] [file...]
```

DESCRIPTION

The *more* utility shall read files and either write them to the terminal on a page-by-page basis or filter them to standard output. If standard output is not a terminal device, all input files shall be copied to standard output in their entirety, without modification, except as specified for the **–s** option. If standard output is a terminal device, the files shall be written a number of lines (one screenful) at a time under the control of user commands. See the EXTENDED DESCRIPTION section.

Certain block-mode terminals do not have all the capabilities necessary to support the complete *more* definition; they are incapable of accepting commands that are not terminated with a <newline>. Implementations that support such terminals shall provide an operating mode to *more* in which all commands can be terminated with a <newline> on those terminals. This mode:

- Shall be documented in the system documentation
- Shall, at invocation, inform the user of the terminal deficiency that requires the <newline> usage and provide instructions on how this warning can be suppressed in future invocations
- Shall not be required for implementations supporting only fully capable terminals
- Shall not affect commands already requiring <newline> characters
- Shall not affect users on the capable terminals from using *more* as described in this volume of POSIX.1-2024

OPTIONS

The *more* utility shall conform to XBD Section 12.2 (on page 215), except that `'+'` may be recognized as an option delimiter as well as `'–'`.

The following options shall be supported:

–c If a screen is to be written that has no lines in common with the current screen, or *more* is writing its first screen, *more* shall not scroll the screen, but instead shall redraw each line of the screen in turn, from the top of the screen to the bottom. In addition, if *more* is writing its first screen, the screen shall be cleared. This option may be silently ignored on devices with insufficient terminal capabilities.

–e Exit immediately after writing the last line of the last file in the argument list; see the EXTENDED DESCRIPTION section.

–i Perform pattern matching in a case-insensitive manner; see XBD Section 9.2 (on page 180).

–n *number* Specify the number of lines per screenful. The *number* argument is a positive decimal integer. The **–n** option shall override any values obtained from any other source.

–p *command* Each time a screen from a new file is displayed or redisplayed (including as a result of *more* commands; for example, **:p**), execute the *more* command(s) in the command arguments in the order specified, as if entered by the user after the first screen has been displayed. No intermediate results shall be displayed (that is, if the command is a movement to a screen different from the normal first screen, only the

screen resulting from the command shall be displayed.) If any of the commands fail for any reason, an informational message to this effect shall be written, and no further commands specified using the −**p** option shall be executed for this file.

−**s** Behave as if consecutive empty lines were a single empty line.

−**t** *tagstring* Write the screenful of the file containing the tag named by the *tagstring* argument. See the *ctags* utility. The tags feature represented by −**t** *tagstring* and the **:t** command is optional. It shall be provided on any system that also provides a conforming implementation of *ctags*; otherwise, the use of −**t** produces undefined results.

The filename resulting from the −**t** option shall be logically added as a prefix to the list of command line files, as if specified by the user. If the tag named by the *tagstring* argument is not found, it shall be an error, and *more* shall take no further action.

If the tag specifies a line number, the first line of the display shall contain the beginning of that line. If the tag specifies a pattern, the first line of the display shall contain the beginning of the matching text from the first line of the file that contains that pattern. If the line does not exist in the file or matching text is not found, an informational message to this effect shall be displayed, and *more* shall display the default screen as if −**t** had not been specified.

If both the −**t** *tagstring* and −**p** *command* options are given, the −**t** *tagstring* shall be processed first; that is, the file and starting line for the display shall be as specified by −**t**, and then the −**p** *more* command shall be executed. If the line (matching text) specified by the −**t** command does not exist (is not found), no −**p** *more* command shall be executed for this file at any time.

−**u** Treat a <backspace> as a printable control character, displayed as an implementation-defined character sequence (see the EXTENDED DESCRIPTION section), suppressing backspacing and the special handling that produces underlined or standout mode text on some terminal types. Also, do not ignore a <carriage-return> at the end of a line.

OPERANDS

The following operand shall be supported:

file A pathname of an input file. If no *file* operands are specified, the standard input shall be used. If a *file* is `'−'`, the standard input shall be read at that point in the sequence.

STDIN

The standard input shall be used only if no *file* operands are specified, or if a *file* operand is `'−'`.

INPUT FILES

The input files being examined shall be text files. If standard output is a terminal, standard error shall be used to read commands from the user. If standard output is a terminal, standard error is not readable, and command input is needed, *more* may attempt to obtain user commands from the controlling terminal (for example, **/dev/tty**); otherwise, *more* shall terminate with an error indicating that it was unable to read user commands. If standard output is not a terminal, no error shall result if standard error cannot be opened for reading.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *more*:

COLUMNS Override the system-selected horizontal display line size. See XBD Chapter 8 (on page 167) for valid values and results when it is unset or null.

EDITOR Used by the **v** command to select an editor. See the EXTENDED DESCRIPTION section.

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes within regular expressions.

LC_MESSAGES Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

LINES Override the system-selected vertical screen size, used as the number of lines in a screenful. See XBD Chapter 8 (on page 167) for valid values and results when it is unset or null. The **−n** option shall take precedence over the *LINES* variable for determining the number of lines in a screenful.

MORE Determine a string containing options described in the OPTIONS section preceded with <hyphen-minus> characters and <blank>-separated as on the command line. Any command line options shall be processed after those in the *MORE* variable, as if the command line were:

```
more $MORE options operands
```

The *MORE* variable shall take precedence over the *TERM* and *LINES* variables for determining the number of lines in a screenful.

TERM Determine the name of the terminal type. If this variable is unset or null, an unspecified default terminal type is used.

ASYNCHRONOUS EVENTS

The following actions shall be taken upon receipt of signals:

SIGCONT The actions described below for SIGWINCH shall be taken, except that the screen shall always be refreshed (regardless of whether the terminal window size changed).

SIGWINCH If standard output is a terminal, the current terminal window size associated with the terminal on standard output shall be obtained, as if by a call to XSH *tcgetwinsize*( ). If the terminal window size is successfully obtained, it shall be used

as follows:

- If the *COLUMNS* environment variable is unset or does not contain a number, the horizontal display line size shall be set to the number of columns in the obtained terminal window size.
- If the **–n** option was not specified (neither on the command line nor via the *MORE* environment variable) and the *LINES* environment variable is unset or does not contain a number, the vertical screen size shall be set to the number of rows in the obtained terminal window size.

If the above resulted in either the vertical screen size or the horizontal display line size (or both) changing to a different value, the number of lines available per screen and the number of columns available per line shall be updated correspondingly (see XBD Chapter 8, on page 167) and the screen shall be refreshed; otherwise, the screen may be refreshed.

The action taken for all other signals shall be the default.

STDOUT

The standard output shall be used to write the contents of the input files.

STDERR

The standard error shall be used for diagnostic messages and user commands (see the INPUT FILES section), and, if standard output is a terminal device, to write a prompting string. The prompting string shall appear on the screen line below the last line of the file displayed in the current screenful. The prompt shall contain the name of the file currently being examined and shall contain an end-of-file indication and the name of the next file, if any, when prompting at the end-of-file. If an error or informational message is displayed, it is unspecified whether it is contained in the prompt. If it is not contained in the prompt, it shall be displayed and then the user shall be prompted for a continuation character, at which point another message or the user prompt may be displayed. The prompt is otherwise unspecified. It is unspecified whether informational messages are written for other user commands.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

The following section describes the behavior of *more* when the standard output is a terminal device. If the standard output is not a terminal device, no options other than **–s** shall have any effect, and all input files shall be copied to standard output otherwise unmodified, at which time *more* shall exit without further action.

The number of lines available per screen shall be determined by the **–n** option, if present, or by obtaining the vertical screen size from the *LINES* environment variable (see the ENVIRONMENT VARIABLES section) or from the terminal window size associated with the terminal on standard output (see XSH *tcgetwinsize*( )), with a default value as described in XBD Chapter 8 (on page 167).

The maximum number of lines written shall be one less than this number, because the screen line after the last line written shall be used to write a user prompt and user input. If the number of lines in the screen is less than two, the results are undefined. It is unspecified whether user input is permitted to be longer than the remainder of the single line where the prompt has been written.

The number of columns available per line shall be determined by obtaining the horizontal display line size from the *COLUMNS* environment variable (see the ENVIRONMENT

VARIABLES section) or from the terminal window size associated with the terminal on standard output (see XSH *tcgetwinsize*( )), with a default value as described in XBD Chapter 8 (on page 167).

Lines that are longer than the display shall be folded; the length at which folding occurs is unspecified, but should be appropriate for the output device. Folding may occur between glyphs of single characters that take up multiple display columns.

When standard output is a terminal and **−u** is not specified, *more* shall treat <backspace> and <carriage-return> characters specially:

- A character, followed first by a sequence of *n* <backspace> characters (where *n* is the same as the number of column positions that the character occupies), then by *n* <underscore> characters (`'_'`), shall cause that character to be written as underlined text, if the terminal type supports that. The *n* <underscore> characters, followed first by *n* <backspace> characters, then any character with *n* column positions, shall also cause that character to be written as underlined text, if the terminal type supports that.
- A sequence of *n* <backspace> characters (where *n* is the same as the number of column positions that the previous character occupies) that appears between two identical printable characters shall cause the first of those two characters to be written as emboldened text (that is, visually brighter, standout mode, or inverse-video mode), if the terminal type supports that, and the second to be discarded. Immediately subsequent occurrences of <backspace>/character pairs for that same character shall also be discarded. (For example, the sequence `"a\ba\ba\ba"` is interpreted as a single emboldened `'a'`.)
- The *more* utility shall logically discard all other <backspace> characters from the line as well as the character which precedes them, if any.
- A <carriage-return> at the end of a line shall be ignored, rather than being written as a non-printable character, as described in the next paragraph.

It is implementation-defined how other non-printable characters are written. Implementations should use the same format that they use for the *ex* **print** command; see the OPTIONS section within the *ed* utility. It is unspecified whether a multi-column character shall be separated if it crosses a display line boundary; it shall not be discarded. The behavior is unspecified if the number of columns on the display is less than the number of columns any single character in the line being displayed would occupy.

When each new file is displayed (or redisplayed), *more* shall write the first screen of the file. Once the initial screen has been written, *more* shall prompt for a user command. If the execution of the user command results in a screen that has lines in common with the current screen, and the device has sufficient terminal capabilities, *more* shall scroll the screen; otherwise, it is unspecified whether the screen is scrolled or redrawn.

For all files but the last (including standard input if no file was specified, and for the last file as well, if the **−e** option was not specified), when *more* has written the last line in the file, *more* shall prompt for a user command. This prompt shall contain the name of the next file as well as an indication that *more* has reached end-of-file. If the user command is **f**, <control>-F, <space>, **j**, <newline>, **d**, <control>-D, or **s**, *more* shall display the next file. Otherwise, if displaying the last file, *more* shall exit. Otherwise, *more* shall execute the user command specified.

Several of the commands described in this section display a previous screen from the input stream. In the case that text is being taken from a non-rewindable stream, such as a pipe, it is implementation-defined how much backwards motion is supported. If a command cannot be executed because of a limitation on backwards motion, an error message to this effect shall be

displayed, the current screen shall not change, and the user shall be prompted for another command.

If a command cannot be performed because there are insufficient lines to display, *more* shall alert the terminal. If a command cannot be performed because there are insufficient lines to display or a **/** command fails: if the input is the standard input, the last screen in the file may be displayed; otherwise, the current file and screen shall not change, and the user shall be prompted for another command.

The interactive commands in the following sections shall be supported. Some commands can be preceded by a decimal integer, called *count* in the following descriptions. If not specified with the command, *count* shall default to 1. In the following descriptions, *pattern* is a basic regular expression, as described in XBD Section 9.3 (on page 181). The term ``examine'' is historical usage meaning ``open the file for viewing''; for example, *more* **foo** would be expressed as examining file **foo**.

In the following descriptions, unless otherwise specified, *line* is a line in the *more* display, not a line from the file being examined.

In the following descriptions, the *current position* refers to two things:

1. The position of the current line on the screen
2. The line number (in the file) of the current line on the screen

Usually, the line on the screen corresponding to the current position is the third line on the screen. If this is not possible (there are fewer than three lines to display or this is the first page of the file, or it is the last page of the file), then the current position is either the first or last line on the screen as described later.

Help

Synopsis: `h`

Write a summary of these commands and other implementation-defined commands. The behavior shall be as if the *more* utility were executed with the **–e** option on a file that contained the summary information. The user shall be prompted as described earlier in this section when end-of-file is reached. If the user command is one of those specified to continue to the next file, *more* shall return to the file and screen state from which the **h** command was executed.

Scroll Forward One Screenful

Synopsis: **[***count***]**`f`
[*count***]**`<control>-F`

Scroll forward *count* lines, with a default of one screenful. If *count* is more than the screen size, only the final screenful shall be written.

Scroll Backward One Screenful

Synopsis: **[***count***]**`b`
[*count***]**`<control>-B`

Scroll backward *count* lines, with a default of one screenful (see the **–n** option). If *count* is more than the screen size, only the final screenful shall be written.

107510 **Scroll Forward One Line**

107511 *Synopsis*: **[***count***]**<space>
107512 **[***count***]**j
107513 **[***count***]**<newline>

107514 Scroll forward *count* lines. The default *count* for the <space> shall be one screenful; for **j** and
107515 <newline>, one line. The entire *count* lines shall be written, even if *count* is more than the screen
107516 size.

107517 **Scroll Backward One Line**

107518 *Synopsis*: **[***count***]**k

107519 Scroll backward *count* lines. The entire *count* lines shall be written, even if *count* is more than the
107520 screen size.

107521 **Scroll Forward One Half Screenful**

107522 *Synopsis*: **[***count***]**d
107523 **[***count***]**<control>–D

107524 Scroll forward *count* lines, with a default of one half of the screen size. If *count* is specified, it
107525 shall become the new default for subsequent **d**, <control>-D, and **u** commands.

107526 **Skip Forward One Line**

107527 *Synopsis*: **[***count***]**s

107528 Display the screenful beginning with the line *count* lines after the last line on the current screen.
107529 If *count* would cause the current position to be such that less than one screenful would be
107530 written, the last screenful in the file shall be written.

107531 **Scroll Backward One Half Screenful**

107532 *Synopsis*: **[***count***]**u
107533 **[***count***]**<control>–U

107534 Scroll backward *count* lines, with a default of one half of the screen size. If *count* is specified, it
107535 shall become the new default for subsequent **d**, <control>–D, **u**, and <control>–U commands.
107536 The entire *count* lines shall be written, even if *count* is more than the screen size.

107537 **Go to Beginning of File**

107538 *Synopsis*: **[***count***]**g

107539 Display the screenful beginning with line *count*.

107540 **Go to End-of-File**

107541 *Synopsis*: **[***count***]**G

107542 If *count* is specified, display the screenful beginning with the line *count*. Otherwise, display the
107543 last screenful of the file.

107544 **Refresh the Screen**

107545 *Synopsis*: `r`
107546 `<control>-L`

107547 Refresh the screen.

107548 **Discard and Refresh**

107549 *Synopsis*: `R`

107550 Refresh the screen, discarding any buffered input. If the current file is non-seekable, buffered
107551 input shall not be discarded and the **R** command shall be equivalent to the **r** command.

107552 **Mark Position**

107553 *Synopsis*: `m`*`letter`*

107554 Mark the current position with the letter named by *letter*, where *letter* represents the name of one
107555 of the lowercase letters of the portable character set. When a new file is examined, all marks may
107556 be lost.

107557 **Return to Mark**

107558 *Synopsis*: `'`*`letter`*

107559 Return to the position that was previously marked with the letter named by *letter*, making that
107560 line the current position.

107561 **Return to Previous Position**

107562 *Synopsis*: `''`

107563 Return to the position from which the last large movement command was executed (where a
107564 ``large movement'' is defined as any movement of more than a screenful of lines). If no such
107565 movements have been made, return to the beginning of the file.

107566 **Search Forward for Pattern**

107567 *Synopsis*: `[`*`count`*`]/[!]`*`pattern`*`<newline>`

107568 Display the screenful beginning with the *count*th line containing the pattern. The search shall
107569 start after the first line currently displayed. The null regular expression (`'/'` followed by a
107570 **<**newline**>**) shall repeat the search using the previous regular expression, with a default *count*. If
107571 the character `'!'` is included, the matching lines shall be those that do not contain the *pattern*. If
107572 no match is found for the *pattern*, a message to that effect shall be displayed.

107573 **Search Backward for Pattern**

107574 *Synopsis*: `[`*`count`*`]?[!]`*`pattern`*`<newline>`

107575 Display the screenful beginning with the *count*th previous line containing the pattern. The search
107576 shall start on the last line before the first line currently displayed. The null regular expression
107577 (`'?'` followed by a **<**newline**>**) shall repeat the search using the previous regular expression,
107578 with a default *count*. If the character `'!'` is included, matching lines shall be those that do not
107579 contain the *pattern*. If no match is found for the *pattern*, a message to that effect shall be
107580 displayed.

107581 **Repeat Search**

107582 *Synopsis*: **[***count***]**n

107583 Repeat the previous search for *count*th line containing the last *pattern* (or not containing the last
107584 *pattern*, if the previous search was "/!" or "?!").

107585 **Repeat Search in Reverse**

107586 *Synopsis*: **[***count***]**N

107587 Repeat the search in the opposite direction of the previous search for the *count*th line containing
107588 the last *pattern* (or not containing the last *pattern*, if the previous search was "/!" or "?!").

107589 **Examine New File**

107590 *Synopsis*: **:e [***filename***]**<newline>

107591 Examine a new file. If the *filename* argument is not specified, the current file (see the **:n** and **:p**
107592 commands below) shall be re-examined. The *filename* shall be subjected to the process of shell
107593 word expansions (see Section 2.6, on page 2483); if more than a single pathname results, the
107594 effects are unspecified. If *filename* is a <number-sign> ('#'), the previously examined file shall
107595 be re-examined. If *filename* is not accessible for any reason (including that it is a non-seekable
107596 file), an error message to this effect shall be displayed and the current file and screen shall not
107597 change.

107598 **Examine Next File**

107599 *Synopsis*: **[***count***]:**n

107600 Examine the next file. If a number *count* is specified, the *count*th next file shall be examined. If
107601 *filename* refers to a non-seekable file, the results are unspecified.

107602 **Examine Previous File**

107603 *Synopsis*: **[***count***]:**p

107604 Examine the previous file. If a number *count* is specified, the *count*th previous file shall be
107605 examined. If *filename* refers to a non-seekable file, the results are unspecified.

107606 **Go to Tag**

107607 *Synopsis*: **:t** *tagstring*<newline>

107608 If the file containing the tag named by the *tagstring* argument is not the current file, examine the
107609 file, as if the **:e** command was executed with that file as the argument. Otherwise, or in addition,
107610 display the screenful beginning with the tag, as described for the **–t** option (see the OPTIONS
107611 section). If the *ctags* utility is not supported by the system, the use of **:t** produces undefined
107612 results.

107613 **Invoke Editor**

107614 *Synopsis*: `v`

107615 Invoke an editor to edit the current file being examined. If standard input is being examined, the
107616 results are unspecified. The name of the editor shall be taken from the environment variable
107617 *EDITOR*, or shall default to *vi*. If the last pathname component in *EDITOR* is either *vi* or *ex*, the
107618 editor shall be invoked with a **−c** *linenumber* command line argument, where *linenumber* is the
107619 line number of the file line containing the display line currently displayed as the first line of the
107620 screen. It is implementation-defined whether line-setting options are passed to editors other
107621 than *vi* and *ex*.

107622 When the editor exits, *more* shall resume with the same file and screen as when the editor was
107623 invoked.

107624 **Display Position**

107625 *Synopsis*: `=`
107626 `<control>-G`

107627 Write a message for which the information references the first byte of the line after the last line of
107628 the file on the screen. This message shall include the name of the file currently being examined,
107629 its number relative to the total number of files there are to examine, the line number in the file,
107630 the byte number and the total bytes in the file, and what percentage of the file precedes the
107631 current position. If *more* is reading from standard input, or the file is shorter than a single screen,
107632 the line number, the byte number, the total bytes, and the percentage need not be written.

107633 **Quit**

107634 *Synopsis*: `q`
107635 `:q`
107636 `ZZ`

107637 Exit *more*.

107638 **EXIT STATUS**

107639 The following exit values shall be returned:

107640 0 Successful completion.

107641 >0 An error occurred.

107642 **CONSEQUENCES OF ERRORS**

107643 If an error is encountered accessing a file when using the **:n** command, *more* shall attempt to
107644 examine the next file in the argument list, but the final exit status shall be affected. If an error is
107645 encountered accessing a file via the **:p** command, *more* shall attempt to examine the previous file
107646 in the argument list, but the final exit status shall be affected. If an error is encountered accessing
107647 a file via the **:e** command, *more* shall remain in the current file and the final exit status shall not
107648 be affected.

APPLICATION USAGE

When the standard output is not a terminal, only the **−s** filter-modification option is effective. This is based on historical practice. For example, a typical implementation of *man* pipes its output through *more* **−s** to squeeze excess white space for terminal users. When *man* is piped to *lp*, however, it is undesirable for this squeezing to happen.

EXAMPLES

The **−p** allows arbitrary commands to be executed at the start of each file. Examples are:

more **−p G** *file1 file2*
Examine each file starting with its last screenful.

more **−p** 100 *file1 file2*
Examine each file starting with line 100 in the current position (usually the third line, so line 98 would be the first line written).

more **−p** /100 *file1 file2*
Examine each file starting with the first line containing the string `"100"` in the current position

RATIONALE

The *more* utility, available in BSD and BSD-derived systems, was chosen as the prototype for the POSIX file display program since it is more widely available than either the public-domain program *less* or than *pg*, a pager provided in System V. The 4.4 BSD *more* is the model for the features selected; it is almost fully upwards-compatible from the 4.3 BSD version in wide use and has become more amenable for *vi* users. Several features originally derived from various file editors, found in both *less* and *pg*, have been added to this volume of POSIX.1-2024 as they have proved extremely popular with users.

There are inconsistencies between *more* and *vi* that result from historical practice. For example, the single-character commands **h**, **f**, **b**, and <space> are screen movers in *more*, but cursor movers in *vi*. These inconsistencies were maintained because the cursor movements are not applicable to *more* and the powerful functionality achieved without the use of the control key justifies the differences.

The tags interface has been included in a program that is not a text editor because it promotes another degree of consistent operation with *vi*. It is conceivable that the paging environment of *more* would be superior for browsing source code files in some circumstances.

The operating mode referred to for block-mode terminals effectively adds a <newline> to each Synopsis line that currently has none. So, for example, **d**<newline> would page one screenful. The mode could be triggered by a command line option, environment variable, or some other method. The details are not imposed by this volume of POSIX.1-2024 because there are so few systems known to support such terminals. Nevertheless, it was considered that all systems should be able to support *more* given the exception cited for this small community of terminals because, in comparison to *vi*, the cursor movements are few and the command set relatively amenable to the optional <newline> characters.

Historically some versions of *more* did not obtain the terminal window size on receipt of SIGCONT, resulting in incorrect screen contents when the screen was refreshed if the size had been changed while *more* was suspended. This is considered to be a bug in those implementations.

Some versions of *more* provide a shell escaping mechanism similar to the *ex* **!** command. The standard developers did not consider that this was necessary in a paginator, particularly given the wide acceptance of multiple window terminals and job control features. (They chose to retain such features in the editors and *mailx* because the shell interaction also gives an

opportunity to modify the editing buffer, which is not applicable to *more*.)

The **−p** (position) option replaces the **+** command because of the Utility Syntax Guidelines. The *+command* option is no longer specified by POSIX.1-2024 but may be present in some implementations. In early proposals, it took a *pattern* argument, but historical *less* provided the *more* general facility of a command. It would have been desirable to use the same **−c** as *ex* and *vi*, but the letter was already in use.

The text stating ``from a non-rewindable stream ... implementations may limit the amount of backwards motion supported'' would allow an implementation that permitted no backwards motion beyond text already on the screen. It was not possible to require a minimum amount of backwards motion that would be effective for all conceivable device types. The implementation should allow the user to back up as far as possible, within device and reasonable memory allocation constraints.

Historically, non-printable characters were displayed using the ARPA standard mappings, which are as follows:

1. Printable characters are left alone.
2. Control characters less than \177 are represented as followed by the character offset from the `'@'` character in the ASCII map; for example, \007 is represented as `'G'`.
3. \177 is represented as followed by `'?'`.

The display of characters having their eighth bit set was less standard. Existing implementations use hex (0x00), octal (\000), and a meta-bit display. (The latter displayed characters with their eighth bit set as the two characters `"M-"`, followed by the seven-bit display as described previously.) The latter probably has the best claim to historical practice because it was used with the **−v** option of 4 BSD and 4 BSD-derived versions of the *cat* utility since 1980.

No specific display format is required by POSIX.1-2024. Implementations are encouraged to conform to historic practice in the absence of any strong reason to diverge.

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 2 (on page 2472), *ctags*, *ed*, *ex*, *vi*

XBD Chapter 8 (on page 167), Section 9.2 (on page 180), Section 9.3 (on page 181), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The obsolescent SYNOPSIS is removed.

The utility has been extensively reworked for alignment with the IEEE P1003.2b draft standard:

- Changes have been made as a result of IEEE PASC Interpretations 1003.2 #37 and #109.
- The *more* utility should be able to handle underlined and emboldened displays of characters that are wider than a single column position.

Issue 7

Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that `'+'` may be recognized as an option delimiter in the OPTIONS section.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0123 [265] is applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0146 [584] is applied.

Issue 8

Austin Group Defect 1031 is applied, changing the description of the **–i** option.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1185 is applied, changing the ASYNCHRONOUS EVENTS and EXTENDED DESCRIPTION sections in relation to the terminal window size.

NAME

msgfmt — create messages objects from portable messages object source files

SYNOPSIS

```
msgfmt [-cfSv] [-D dir] [-o outputfile] pathname...
```

DESCRIPTION

The *msgfmt* utility shall create messages object files from portable messages object source files (dot-po files).

A dot-po file contains messages to be output by system commands or by applications. The messages in these files should be able to be translated to any language supported by the system.

The *msgfmt* utility shall interpret message strings for output as characters according to the codeset specified in the dot-po file or, if not present, the current setting of the *LC_CTYPE* locale category.

OPTIONS

The *msgfmt* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−c If this option and **−v** are both specified, *msgfmt* shall detect and diagnose input file abnormalities which might represent translation errors. The **msgid** and **msgstr** strings shall be compared. It shall be considered abnormal if one string starts or ends with a <newline> while the other does not. Also, if the flag **c-format** appears in a `"#,"` comment for a **msgid** directive (see EXTENDED DESCRIPTION), it shall be considered abnormal if the strings do not have the same number of `'%'` conversion specifiers, or if corresponding conversion specifiers take different argument types (see XSH *fprintf( )*, on page 995). If an abnormality is detected, the exit status shall be non-zero and a diagnostic message shall be output. Additional checks beyond those described here may also be performed. These checks may produce diagnostics or informational messages and need not affect the exit status. If **−c** is specified without **−v** or **−v** is specified without **−c**, the behavior is unspecified.

−D *dir* Add *dir* to the list of directories to search for input files.

−f Use fuzzy entries in output. If this option is not specified, fuzzy entries shall not be included in the output.

−o *outputfile*
Specify the name of an output file to be used instead of the default filename(s) specified in EXTENDED DESCRIPTION. All **domain** *domainname* directives in the dot-po file(s) shall be ignored.

−S Append the suffix **.mo** to each generated messages object filename if it does not have this suffix.

−v See **−c**.

OPERANDS

The following operand shall be supported:

pathname A pathname of a dot-po file.

STDIN

Not used.

INPUT FILES

The input files shall be text files in the format described in EXTENDED DESCRIPTION.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *msgfmt*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

XSI *LANGUAGE* Determine the location of messages objects if *NLSPATH* is not set or the evaluation of *NLSPATH* did not lead to a suitable messages object being found.

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale name used to locate messages objects, and the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not Used.

STDERR

The standard error shall be used for diagnostic messages and may also be used for warning messages. If the **−c** and **−v** options are specified, additional unspecified informational messages may be written to standard error.

OUTPUT FILES

The format of the created messages object files is unspecified.

EXTENDED DESCRIPTION

The *msgfmt* utility shall accept portable messages object source files (dot-po files) in the following format.

A dot-po file contains zero or more lines, with each non-blank line containing a comment, a statement, or a statement continuation. A comment has an unquoted <number-sign> (`'#'`) as the first non-<blank> character and ends with the next <newline> character. A statement continuation is a double-quoted string on a line by itself, optionally preceded and/or followed by <blank> characters, and the string shall be concatenated with the string on the previous statement line. If a comment occurs between a statement and a statement continuation, the behavior is unspecified. All other comments, except for comments beginning with <number-sign><comma> (`"#,"`), and blank lines shall be ignored.

The format of a statement is:

```
directive value
```


The *directive* starts at the first non-<blank> character of the line and is separated from the *value* by one or more <blank> characters. The *value* consists of a double-quoted string optionally followed by <blank> characters. Zero or more statement continuation lines (see above) can follow the statement. The following directives shall be supported:

domain *domainname*
msgid *message_identifier*
msgid_plural *untranslated_string_plural*
msgstr *message_string*
msgstr[*index***]** *message_string*

A dot-po file consists of zero or more sections. Each section specifies the messages to be processed in a domain. The first directive in each section shall be a **domain** directive (except for the first section which shall behave as if

```
domain "messages"
```

had been specified if the first directive is not a **domain** directive).

The behavior of the **domain** directive is affected by the options used. See OPTIONS for the behavior when the **−o** option is specified. If the **−o** option is not specified, all data obtained from the non-**domain** directives in a dot-po section shall be output to the messages object file named *domainname***.mo** when the **−S** option is specified. When the **−S** option is not specified, it is implementation-defined whether *domainname* or *domainname***.mo** is used.

If multiple **domain** directives specify the same *domainname*, the sections shall be processed as if there was only one section that starts with a **domain** *domainname* statement which contained the statements of the sections, in the same order, excluding all but the first **domain** *domainname* statement.

Within each section, there can be a header. A header is identified by having a **msgid** directive with the empty string (**""**) as the *message_identifier* immediately followed by a statement containing a **msgstr** directive. The *message_string* in this **msgstr** statement in a header shall be treated specially. If *message_string* contains a specification of the form:

```
"nplurals=count; plural=expression"
```

then *count* indicates the number of plural forms for messages in that domain, and *expression* is a C-language expression that evaluates to an unsigned integer value which determines the **msgstr[***index***]** directive to be used. The value of *expression* is used as the index value. The variable *n* in *expression* is assigned the value of the *n* argument to the *ngettext*(), *ngettext_l*(), *dngettext*(), *dngettext_l*(), *dcngettext*(), and *dcngettext_l*() functions or of the *n* operand of the *ngettext* utility before *expression* is evaluated. The application shall ensure that *expression* evaluates to a non-negative value less than *count* for all *n* that can be supplied by the aforementioned functions and utility.

If *message_string* in the header contains a specification of the form:

```
"charset=codeset"
```

then *codeset* indicates the codeset to be used to encode the message strings in this section's domain (overriding *LC_CTYPE*). If the output string's codeset is different from the message string's codeset, codeset conversion from the message string's codeset to the output string's codeset shall be performed by the *gettext* family of functions and by the *gettext* and *ngettext* utilities. See XSH *gettext* and *gettext*. The output string's codeset shall be determined by the current or specified locale's codeset.

Note: It is the responsibility of translators to ensure that the characters they enter into message strings in a dot-po file are encoded in the codeset specified in the header.

If a header is present in a section, the application shall ensure that the header is provided by the first **msgid** directive in that section.

After the header, if present, zero or more messages are identified by a **msgid** directive with a *message_identifier* that is not an empty string. Each of these directives start a subsection that is used to get a translated message from the *gettext* family of functions and from the *gettext* and *ngettext* utilities. If the *message_identifier* string is the string identified by the *gettext* family of functions *msgid* argument or by the *gettext* and *ngettext* utility *msgid* operand, this subsection specifies how that translation is to be processed.

If there is only a singular form for the given *message_identifier*, the application shall ensure that the statement containing the **msgid** directive is immediately followed by a **msgstr** directive.

If there are plural forms for the given *message_identifier* and the header for this section exists and contains an

`"nplurals=`*count*`; plural=`*expression*`"`

specification, the application shall ensure that the statement containing the **msgid** directive is immediately followed by a **msgid_plural** directive and that each statement containing a **msgid_plural** directive is followed by *count* statements containing **msgstr[***index***]** directives, starting with **msgstr[0]** and ending with **msgstr[***count***–1]** in increasing order, with no duplicate index values. If a header for this section does not exist or does not contain an

`"nplurals=`*count*`; plural=`*expression*`"`

specification, the application shall ensure that no **msgid_plural** or **msgstr[***index***]** directives are used in this section.

For example, if the header's *message_string* contains the specification:

```
"nplurals=2; plural= n == 1 ? 0 : 1"
```

there are two forms in the domain; **msgstr[0]** is used if *n* is equal to 1, otherwise **msgstr[1]** is used. For another example, if the header's *message_string* contains:

```
"nplurals=3; plural= n == 1 ? 0 : n == 2 ? 1 : 2"
```

there are three forms in the domain; **msgstr[0]** is used if *n* is equal to 1, **msgstr[1]** is used if *n* is equal to 2, otherwise **msgstr[2]** is used.

C-language escape sequences in strings shall be processed as specified for character string literals in the ISO C standard, except that *universal-character-name* escape sequences need not be supported.

Comments in a dot-po file can be in one of the following formats:

`#: `*reference*
`#. `*utility-added-comments*
`#, `*flag*
`#`*translator-comments* (where *translator-comments* does not begin with `'.'`, `':'` or `','`)

A `#: `*reference* comment indicates the location(s) of the **msgid** string in the source files, in

pathname1`:`*linenumber1* `[`*pathname2*`:`*linenumber2* `... ]`

format. They can be added, as might `"#."` prefixed additional comments of unspecified format, by the *xgettext* utility. All comments that do not begin with `"#,"` are informative only and shall be silently ignored by the *msgfmt* utility. In `"#,"` comments the following values for *flag* can be

107921 specified:

107922 **fuzzy** This flag indicates that the **msgstr** string might not be a correct translation at this
107923 point in time. Only the translator can judge if the translation requires further
107924 modification or is acceptable as is. Once satisfied with the translation, the
107925 translator should remove this **fuzzy** flag. If this flag is specified, the *msgfmt* utility
107926 shall not generate the entry for the next following **msgid** in the output message
107927 catalog, unless the **−f** option is specified. If other flag comments are specified
107928 between **fuzzy** and the **msgid**, the behavior is unspecified.

107929 **c-format**
107930 **no-c-format** The **c-format** flag indicates that the next following **msgid** string contains a *printf*( )
107931 format string. When the **c-format** flag is given and the **−c** and **−v** options are
107932 specified, the *msgfmt* utility shall perform additional tests to check the validity of
107933 the translation (see OPTIONS); these additional tests may also be performed if
107934 neither **c-format** nor **no-c-format** is given. When the **no-c-format** flag is given for a
107935 string, no additional checks shall be performed for the string. When both the **c-**
107936 **format** and the **no-c-format** flags are given, the last flag specified takes precedence.

107937 **EXIT STATUS**

107938 The following exit values shall be returned:

107939 0 Successful completion.

107940 >0 An error occurred.

107941 **CONSEQUENCES OF ERRORS**

107942 The *msgfmt* utility need not continue processing later *pathname* operands when an error
107943 condition that affects the exit status is detected. It is unspecified whether a messages object file is
107944 written when checks performed for the **−c** and **−v** options fail.

107945 **APPLICATION USAGE**

107946 The *xgettext* utility can be used to create template dot-po files from C-language source files.

107947 Installing messages object files for the POSIX or C locale is not recommended, since they may be
107948 ignored for the sake of efficiency.

107949 The first section for each domain in a dot-po file should include a header containing a

107950 `"charset=`*`codeset`*`"`

107951 specification. If this specification is omitted, message conversions in the *gettext* family of
107952 functions and in the *gettext* and *ngettext* utilities may fail.

107953 The **msgid_plural** directive's *untranslated_string_plural* string comes from the *msgid_plural*
107954 arguments in calls to the *ngettext*( ), *ngettext_l*( ), *dngettext*( ), *dngettext_l*( ), *dcngettext*( ), and
107955 *dcngettext_l*( ) functions when a prototype dot-po file is created by the *xgettext* utility. These
107956 strings (and the *msgid_plural* operands in calls to the *ngettext* utility) can provide context when a
107957 translator is modifying a template dot-po file into a dot-po file for a specific language. These
107958 functions and the *ngettext* utility do not try to match the *msgid_plural* arguments or operands
107959 with anything in a messages object file; they only match the *msgid* arguments and operands.

107960 Unlike shell command language strings, double-quoted strings in dot-po files cannot contain a
107961 literal <newline> character.

EXAMPLES

In this example, **module1.po** and **module2.po** are portable messages object source files.

```
$ cat module1.po
# default domain "messages"
msgid ""
msgstr "charset=utf-8"
msgid "msg 1"
msgstr "msg 1 translation"
#
domain "help_domain"
msgid ""
msgstr "charset=utf-8"
msgid "help 2"
msgstr "help 2 translation"
#
domain "error_domain"
msgid ""
msgstr "charset=utf-8"
msgid "error 3"
msgstr "error 3 translation"
```

```
$ cat module2.po
# default domain "messages"
msgid ""
msgstr "charset=utf-8"
msgid "mesg 4"
msgstr "mesg 4 translation"
#
domain "error_domain"
msgid ""
msgstr "charset=utf-8"
#, c-format
msgid "error 5 %s"
msgstr "error 5 translation %s"
#
domain "window_domain"
msgid ""
msgstr "charset=utf-8"
msgid "window 6"
msgstr "window 6 translation"
```

```
$ cat module3.po
# default domain "messages"
# header will be used for the whole output file in the third example
msgid ""
msgstr "charset=utf-8"
msgid "info 0"
msgstr "info 0 translation"
```

```
$ cat opt_debug.po
#
domain "debug_domain"
msgid "debug 8"
```



```
msgstr "debug 8 translation"
```

The following command will produce the output files **messages.mo**, **help_domain.mo**, and **error_domain.mo**:

```
$ msgfmt -S module1.po
```

The following command will produce the output files **messages.mo**, **help_domain.mo**, **error_domain.mo**, and **window_domain.mo**:

```
$ msgfmt -S module1.po module2.po
```

The following command will produce the output file **hello.mo**:

```
$ msgfmt -o hello.mo module3.po opt_debug.po
```

RATIONALE

Some implementations are less strict about the format of dot-po files and simply treat all occurrences of one or more white space characters as a separator. The format described in this standard is accepted by all known implementations.

In some implementations, duplicate **msgid** directives within a domain are ignored, and only an entry for the first **msgid** directive and the following **msgid**, **msgid_plural**, **msgstr**, or **msgstr**[*index*] directives is created. However, some implementations consider duplicate **msgid** directives within a domain to be an error and do not produce output at all. Consequently this standard does not specify the behavior of *msgfmt* if duplicate **msgid** directives are encountered within one domain.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

gettext, *xgettext*

XSH *fprintf*( ), *gettext*

CHANGE HISTORY

First released in Issue 8.

NAME

mv — move files

SYNOPSIS

```
mv [-if] source_file target_file
mv [-if] source_file... target_dir
```

DESCRIPTION

In the first synopsis form, the *mv* utility shall move the file named by the *source_file* operand to the destination specified by the *target_file*. This first synopsis form is assumed when the final operand does not name an existing directory and is not a symbolic link referring to an existing directory. In this case, if *source_file* names a non-directory file and *target_file* ends with a trailing <slash> character, *mv* shall treat this as an error and no *source_file* operands shall be processed.

In the second synopsis form, *mv* shall move each file named by a *source_file* operand to a destination file in the existing directory named by the *target_dir* operand, or referenced if *target_dir* is a symbolic link referring to an existing directory. The destination path for each *source_file* shall be the concatenation of the target directory, a single <slash> character if the target did not end in a <slash>, and the last pathname component of the *source_file*. This second form is assumed when the final operand names an existing directory.

If any operand specifies an existing file of a type not specified by the System Interfaces volume of POSIX.1-2024, the behavior is implementation-defined.

For each *source_file* the following steps shall be taken:

1. If the destination path exists, the **−f** option is not specified, and either of the following conditions is true:

   a. The permissions of the destination path do not permit writing and the standard input is a terminal.

   b. The **−i** option is specified.

   the *mv* utility shall write a prompt to standard error and read a line from standard input. If the response is not affirmative, *mv* shall do nothing more with the current *source_file* and go on to any remaining *source_file*s.

2. If the *source_file* operand and destination path resolve to either the same existing directory entry or different directory entries for the same existing file, then the destination path shall not be removed, and one of the following shall occur:

   a. No change is made to *source_file*, no error occurs, and no diagnostic is issued.

   b. No change is made to *source_file*, a diagnostic is issued to standard error identifying the two names, and the exit status is affected.

   c. If the *source_file* operand and destination path name distinct directory entries, then the *source_file* operand is removed, no error occurs, and no diagnostic is issued.

   The *mv* utility shall do nothing more with the current *source_file*, and go on to any remaining *source_file*s.

3. The *mv* utility shall perform actions equivalent to the *rename*() function defined in the System Interfaces volume of POSIX.1-2024, called with the following arguments:

   a. The *source_file* operand is used as the *old* argument.

b. The destination path is used as the *new* argument.

If this succeeds, *mv* shall do nothing more with the current *source_file* and go on to any remaining *source_file*s. If this fails for any reasons other than those described for the *errno* [EXDEV] in the System Interfaces volume of POSIX.1-2024, *mv* shall write a diagnostic message to standard error, do nothing more with the current *source_file*, and go on to any remaining *source_file*s.

4. If the destination path exists, and it is a file of type directory and *source_file* is not a file of type directory, or it is a file not of type directory and *source_file* is a file of type directory, *mv* shall write a diagnostic message to standard error, do nothing more with the current *source_file*, and go on to any remaining *source_file*s. If the destination path exists and was created by a previous step, it is unspecified whether this will treated as an error or the destination path will be overwritten.

5. If the destination path exists, *mv* shall attempt to remove it. If this fails for any reason, *mv* shall write a diagnostic message to standard error, do nothing more with the current *source_file*, and go on to any remaining *source_file*s.

6. The file hierarchy rooted in *source_file* shall be duplicated as a file hierarchy rooted in the destination path. If *source_file* or any of the files below it in the hierarchy are symbolic links, the links themselves shall be duplicated, including their contents, rather than any files to which they refer. The following characteristics of each file in the file hierarchy shall be duplicated:

   - The time of last data modification and time of last access
   - The user ID and group ID
   - The file mode

   If the user ID, group ID, or file mode of a regular file cannot be duplicated, the file mode bits S_ISUID and S_ISGID shall not be duplicated.

   When files are duplicated to another file system, the implementation may require that the process invoking *mv* has read access to each file being duplicated.

   If files being duplicated to another file system have hard links to other files, it is unspecified whether the files copied to the new file system have the hard links preserved or separate copies are created for the linked files.

   If the duplication of the file hierarchy fails for any reason, *mv* shall write a diagnostic message to standard error, do nothing more with the current *source_file*, and go on to any remaining *source_file*s.

   If the duplication of the file characteristics fails for any reason, *mv* shall write a diagnostic message to standard error, but this failure shall not cause *mv* to modify its exit status.

7. The file hierarchy rooted in *source_file* shall be removed. If this fails for any reason, *mv* shall write a diagnostic message to the standard error, do nothing more with the current *source_file*, and go on to any remaining *source_file*s.

OPTIONS

The *mv* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–f Do not prompt for confirmation if the destination path exists. Any previous occurrence of the **–i** option is ignored.

–i Prompt for confirmation if the destination path exists. Any previous occurrence of the **–f** option is ignored.

Specifying more than one of the **–f** or **–i** options shall not be considered an error. The last option specified shall determine the behavior of *mv*.

OPERANDS

The following operands shall be supported:

source_file A pathname of a file or directory to be moved.

target_file A new pathname for the file or directory being moved.

target_dir A pathname of an existing directory into which to move the input files.

STDIN

The standard input shall be used to read an input line in response to each prompt specified in the STDERR section. Otherwise, the standard input shall not be used.

INPUT FILES

The input files specified by each *source_file* operand can be of any file type.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *mv*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements used in the extended regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES* category.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes used in the extended regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES* category.

LC_MESSAGES
Determine the locale used to process affirmative responses, and the locale used to affect the format and contents of diagnostic messages and prompts written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

Prompts shall be written to the standard error under the conditions specified in the DESCRIPTION section. The prompts shall contain the destination pathname, but their format is otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic

messages.

OUTPUT FILES

The output files may be of any file type.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All requested files (excluding files where a non-affirmative response was given to a request for confirmation) were successfully moved.

>0 An error occurred.

CONSEQUENCES OF ERRORS

If the copying or removal of *source_file* is prematurely terminated by a signal or error, *mv* may leave a partial copy of *source_file* at the source or destination. The *mv* utility shall not modify both *source_file* and the destination path simultaneously; termination at any point shall leave either *source_file* or the destination path complete.

APPLICATION USAGE

Some implementations mark for update the last file status change timestamp of renamed files and some do not. Applications which make use of the last file status change timestamp may behave differently with respect to renamed files unless they are designed to allow for either behavior.

The specification ensures that *mv* **a a** will not alter the contents of file **a**, and allows the implementation to issue an error that a file cannot be moved onto itself. Likewise, when **a** and **b** are hard links to the same file, *mv* **a b** will not alter **b**, but if a diagnostic is not issued, then it is unspecified whether **a** is left untouched (as it would be by the *rename*( ) function) or unlinked (reducing the link count of **b**).

EXAMPLES

If the current directory contains only files **a** (of any type defined by the System Interfaces volume of POSIX.1-2024), **b** (also of any type), and a directory **c**:

```
mv a b c
mv c d
```

results with the original files **a** and **b** residing in the directory **d** in the current directory.

RATIONALE

Early proposals diverged from the SVID and BSD historical practice in that they required that when the destination path exists, the **−f** option is not specified, and input is not a terminal, *mv* fails. This was done for compatibility with *cp*. The current text returns to historical practice. It should be noted that this is consistent with the *rename*( ) function defined in the System Interfaces volume of POSIX.1-2024, which does not require write permission on the target.

For absolute clarity, paragraph (1), describing the behavior of *mv* when prompting for confirmation, should be interpreted in the following manner:

```
if (exists AND (NOT f_option) AND
    ((not_writable AND input_is_terminal) OR i_option))
```

The **−i** option exists on BSD systems, giving applications and users a way to avoid accidentally unlinking files when moving others. When the standard input is not a terminal, the 4.3 BSD *mv* deletes all existing destination paths without prompting, even when **−i** is specified; this is

inconsistent with the behavior of the 4.3 BSD *cp* utility, which always generates an error when the file is unwritable and the standard input is not a terminal. The standard developers decided that use of **−i** is a request for interaction, so when the destination path exists, the utility takes instructions from whatever responds to standard input.

The *rename*( ) function is able to move directories within the same file system. Some historical versions of *mv* have been able to move directories, but not to a different file system. The standard developers considered that this was an annoying inconsistency, so this volume of POSIX.1-2024 requires directories to be able to be moved even across file systems. There is no **−R** option to confirm that moving a directory is actually intended, since such an option was not required for moving directories in historical practice. Requiring the application to specify it sometimes, depending on the destination, seemed just as inconsistent. The semantics of the *rename*( ) function were preserved as much as possible. For example, *mv* is not permitted to ``rename'' files to or from directories, even though they might be empty and removable.

Historic implementations of *mv* did not exit with a non-zero exit status if they were unable to duplicate any file characteristics when moving a file across file systems, nor did they write a diagnostic message for the user. The former behavior has been preserved to prevent scripts from breaking; a diagnostic message is now required, however, so that users are alerted that the file characteristics have changed.

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified because implementations may desire more descriptive prompts than those used on historical implementations. Therefore, an application not using the **−f** option or using the **−i** option relies on the system to provide the most suitable dialog directly with the user, based on the behavior specified.

When *mv* is dealing with a single file system and *source_file* is a symbolic link, the link itself is moved as a consequence of the dependence on the *rename*( ) functionality, per the DESCRIPTION. Across file systems, this has to be made explicit.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

cp, *ln*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *rename*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

The *mv* utility is changed to describe processing of symbolic links as specified in the IEEE P1003.2b draft standard.

The APPLICATION USAGE section is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #016 is applied.

Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the *LC_MESSAGES* environment variable.

Austin Group Interpretations 1003.1-2001 #164, #168, and #169 are applied.

SD5-XCU-ERN-13 is applied, making an editorial correction to the SYNOPSIS.

SD5-XCU-ERN-51 is applied to the DESCRIPTION, defining the behavior for when files are being duplicated to another file system while having hard links.

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0124 [48] is applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0147 [534] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1732 is applied, changing the EXIT STATUS section.

NAME

newgrp — change to a new group

SYNOPSIS

```
newgrp [-l] [group]
```

DESCRIPTION

The *newgrp* utility shall create a new shell execution environment with a new real and effective group identification. Of the attributes listed in Section 2.13 (on page 2522), the new shell execution environment shall retain the working directory, file creation mask, and exported variables from the previous environment (that is, open files, traps, unexported variables, alias definitions, shell functions, and *set* options may be lost). All other aspects of the process environment that are preserved by the *exec* family of functions defined in the System Interfaces volume of POSIX.1-2024 shall also be preserved by *newgrp*; whether other aspects are preserved is unspecified.

A failure to assign the new group identifications (for example, for security or password-related reasons) shall not prevent the new shell execution environment from being created.

The *newgrp* utility shall affect the supplemental groups for the process as follows:

- On systems where the effective group ID is normally in the supplementary group list (or whenever the old effective group ID actually is in the supplementary group list):
  - If the new effective group ID is also in the supplementary group list, *newgrp* shall change the effective group ID.
  - If the new effective group ID is not in the supplementary group list, *newgrp* shall add the new effective group ID to the list, if there is room to add it.
- On systems where the effective group ID is not normally in the supplementary group list (or whenever the old effective group ID is not in the supplementary group list):
  - If the new effective group ID is in the supplementary group list, *newgrp* shall delete it.
  - If the old effective group ID is not in the supplementary list, *newgrp* shall add it if there is room.

Note: The System Interfaces volume of POSIX.1-2024 does not specify whether the effective group ID of a process is included in its supplementary group list.

With no operands, *newgrp* shall change the effective group back to the groups identified in the user's user entry, and shall set the list of supplementary groups to that set in the user's group database entries.

If the first argument is `'-'`, the results are unspecified.

If a password is required for the specified group, and the user is not listed as a member of that group in the group database, the user shall be prompted to enter the correct password for that group. If the user is listed as a member of that group, no password shall be requested. If no password is required for the specified group, it is implementation-defined whether users not listed as members of that group can change to that group. Whether or not a password is required, implementation-defined system accounting or security mechanisms may impose additional authorization restrictions that may cause *newgrp* to write a diagnostic message and suppress the changing of the group identification.

OPTIONS

The *newgrp* utility shall conform to XBD Section 12.2 (on page 215), except for the unspecified usage of '−'.

The following option shall be supported:

–l (The letter ell.) Change the environment to what would be expected if the user actually logged in again.

OPERANDS

The following operand shall be supported:

group A group name from the group database or a non-negative numeric group ID. Specifies the group ID to which the real and effective group IDs shall be set. If *group* is a non-negative numeric string and exists in the group database as a group name (see *getgrnam*()), the numeric group ID associated with that group name shall be used as the group ID.

STDIN

Not used.

INPUT FILES

The file **/dev/tty** shall be used to read a single line of text for password checking, when one is required.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *newgrp*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used for diagnostic messages and a prompt string for a password, if one is required. Diagnostic messages may be written in cases where the exit status is not available. See the EXIT STATUS section.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

If *newgrp* succeeds in creating a new shell execution environment, whether or not the group identification was changed successfully, the exit status shall be the exit status of the shell. Otherwise, the following exit value shall be returned:

>0 An error occurred.

CONSEQUENCES OF ERRORS

The invoking shell may terminate.

APPLICATION USAGE

There is no convenient way to enter a password into the group database. Use of group passwords is not encouraged, because by their very nature they encourage poor security practices. Group passwords may disappear in the future.

A common implementation of *newgrp* is that the current shell uses *exec* to overlay itself with *newgrp*, which in turn overlays itself with a new shell after changing group. On some implementations, however, this may not occur and *newgrp* may be invoked as a subprocess.

The *newgrp* command is intended only for use from an interactive terminal. It does not offer a useful interface for the support of applications.

The exit status of *newgrp* is generally inapplicable. If *newgrp* is used in a script, in most cases it successfully invokes a new shell and the rest of the original shell script is bypassed when the new shell exits. Used interactively, *newgrp* displays diagnostic messages to indicate problems. But usage such as:

```
newgrp foo
echo $?
```

is not useful because the new shell might not have access to any status *newgrp* may have generated (and most historical systems do not provide this status). A zero status echoed here does not necessarily indicate that the user has changed to the new group successfully. Following *newgrp* with the *id* command provides a portable means of determining whether the group change was successful or not.

EXAMPLES

None.

RATIONALE

Most historical implementations use one of the *exec* functions to implement the behavior of *newgrp*. Errors detected before the *exec* leave the environment unchanged, while errors detected after the *exec* leave the user in a changed environment. While it would be useful to have *newgrp* issue a diagnostic message to tell the user that the environment changed, it would be inappropriate to require this change to some historical implementations.

The password mechanism is allowed in the group database, but how this would be implemented is not specified.

The *newgrp* utility was retained in this volume of POSIX.1-2024, even given the existence of the multiple group permissions feature in the System Interfaces volume of POSIX.1-2024, for several reasons. First, in some implementations, the group ownership of a newly created file is determined by the group of the directory in which the file is created, as allowed by the System

Interfaces volume of POSIX.1-2024; on other implementations, the group ownership of a newly created file is determined by the effective group ID. On implementations of the latter type, *newgrp* allows files to be created with a specific group ownership. Finally, many implementations use the real group ID in accounting, and on such systems, *newgrp* allows the accounting identity of the user to be changed.

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 2 (on page 2472), *sh*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *exec*, *getgrnam*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The obsolescent SYNOPSIS is removed.

The text describing supplemental groups is no longer conditional on {NGROUPS_MAX} being greater than 1. This is because {NGROUPS_MAX} now has a minimum value of 8. This is a FIPS requirement.

Issue 7

Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first argument is `'-'`.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The *newgrp* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

ngettext — retrieve text string from messages object

SYNOPSIS

```
ngettext [-e|-E] [-d textdomain] [textdomain] msgid msgid_plural n
```

DESCRIPTION

Refer to *gettext*.

NAME

nice — invoke a utility with an altered nice value

SYNOPSIS

```
nice [-n increment] utility [argument...]
```

DESCRIPTION

The *nice* utility shall invoke a utility, requesting that it be run with a different nice value (see XBD Section 3.225, on page 64). With no options, the executed utility shall be run with a nice value that is some implementation-defined quantity greater than or equal to the nice value of the current process. If the user lacks appropriate privileges to affect the nice value in the requested manner, the *nice* utility shall not affect the nice value; in this case, a warning message may be written to standard error, but this shall not prevent the invocation of *utility* or affect the exit status.

OPTIONS

The *nice* utility shall conform to XBD Section 12.2 (on page 215).

The following option is supported:

–n *increment* A positive or negative decimal integer which shall have the same effect on the execution of the utility as if the utility had called the *nice*( ) function with the numeric value of the *increment* option-argument.

OPERANDS

The following operands shall be supported:

utility The name of a utility that is to be invoked. If the *utility* operand names any of the special built-in utilities in Section 2.15 (on page 2526), the results are undefined.

argument Any string to be supplied as an argument when invoking the utility named by the *utility* operand.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *nice*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

PATH Determine the search path used to locate the utility to be invoked. See XBD Chapter 8 (on page 167).

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

If *utility* is invoked, the exit status of *nice* shall be the exit status of *utility*; otherwise, the *nice* utility shall exit with one of the following values:

1-125 An error occurred in the *nice* utility.

126 The utility specified by *utility* was found but could not be invoked.

127 The utility specified by *utility* could not be found.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The only guaranteed portable uses of this utility are:

nice utility
Run *utility* with the default higher or equal nice value.

nice –**n** <*positive integer*> *utility*
Run *utility* with a higher nice value.

On some implementations they have no discernible effect on the invoked utility and on some others they are exactly equivalent.

Historical systems have frequently supported the <*positive integer*> up to 20. Since there is no error penalty associated with guessing a number that is too high, users without access to the system conformance document (to see what limits are actually in place) could use the historical 1 to 20 range or attempt to use very large numbers if the job should be truly low priority.

The nice value of a process can be displayed using the command:

```
ps -o nice
```

The *command*, *env*, *nice*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit code 127 if a utility to be invoked cannot be found, so that applications can distinguish ``failure to find a utility'' from ``invoked utility exited with an error indication''. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for ``normal error conditions'' and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any

attempt to *exec* the utility fails for any other reason.

EXAMPLES

None.

RATIONALE

The 4.3 BSD version of *nice* does not check whether *increment* is a valid decimal integer. The command *nice* **–x** *utility*, for example, would be treated the same as the command *nice* **––1** *utility*. If the user does not have appropriate privileges, this results in a ``permission denied'' error. This is considered a bug.

When a user without appropriate privileges gives a negative *increment*, System V treats it like the command *nice* **–0** *utility*, while 4.3 BSD writes a ``permission denied'' message and does not run the utility. The standard specifies the System V behavior together with an optional BSD-style ``permission denied'' message.

The C shell has a built-in version of *nice* that has a different interface from the one described in this volume of POSIX.1-2024.

The term ``utility'' is used, rather than ``command'', to highlight the fact that shell compound commands, pipelines, and so on, cannot be used. Special built-ins also cannot be used. However, ``utility'' includes user application programs and shell scripts, not just utilities defined in this volume of POSIX.1-2024.

Historical implementations of *nice* provide a nice value range of 40 or 41 discrete steps, with the default nice value being the midpoint of that range. By default, they raise the nice value of the executed utility by 10.

Some historical documentation states that the *increment* value must be within a fixed range. This is misleading; the valid *increment* values on any invocation are determined by the current process nice value, which is not always the default.

The definition of nice value is not intended to suggest that all processes in a system have priorities that are comparable. Scheduling policy extensions such as the realtime priorities in the System Interfaces volume of POSIX.1-2024 make the notion of a single underlying priority for all scheduling policies problematic. Some implementations may implement the *nice*-related features to affect all processes on the system, others to affect just the general time-sharing activities implied by this volume of POSIX.1-2024, and others may have no effect at all. Because of the use of ``implementation-defined'' in *nice* and *renice*, a wide range of implementation strategies are possible.

Earlier versions of this standard allowed a –*increment* option. This form is no longer specified by POSIX.1-2024 but may be present in some implementations.

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 2 (on page 2472), *renice*

XBD Section 3.225 (on page 64), Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *nice*( )

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The obsolescent SYNOPSIS is removed.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/18 is applied, deleting a paragraph of RATIONALE that referred to text no longer in the standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #027 is applied.

SD5-XCU-ERN-32 and SD5-XCU-ERN-33 are applied, updating the DESCRIPTION, APPLICATION USAGE, and RATIONALE sections.

The *nice* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1586 is applied, adding the *timeout* utility.

Austin Group Defect 1594 is applied, changing the APPLICATION USAGE section.

NAME

nl — line numbering filter

SYNOPSIS

XSI
```
nl [-p] [-b type] [-d delim] [-f type] [-h type] [-i incr] [-l num]
    [-n format] [-s sep] [-v startnum] [-w width] [file]
```

DESCRIPTION

The *nl* utility shall read lines from the named *file* or the standard input if no *file* is named and shall reproduce the lines to standard output. Lines shall be numbered on the left. Additional functionality may be provided in accordance with the command options in effect.

The *nl* utility views the text it reads in terms of logical pages. Line numbering shall be reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (for example, no numbering of header and footer lines while numbering blank lines only in the body).

The starts of logical page sections shall be signaled by input lines containing nothing but the following delimiter characters:

Line	Start of
`\:\:\:`	Header
`\:\:`	Body
`\:`	Footer

Unless otherwise specified, *nl* shall assume the text being read is in a single logical page body.

OPTIONS

The *nl* utility shall conform to XBD Section 12.2 (on page 215). Only one file can be named.

The following options shall be supported:

–b *type* Specify which logical page body lines shall be numbered. Recognized *types* and their meaning are:

- **a** Number all lines.
- **t** Number only non-empty lines.
- **n** No line numbering.
- **p***string* Number only lines that contain the basic regular expression specified in *string*.

The default *type* for logical page body shall be **t** (text lines numbered).

–d *delim* Specify the delimiter characters that indicate the start of a logical page section. These can be changed from the default characters `"\:"` to two user-specified characters. If only one character is entered, the second character shall remain the default character `':'`.

–f *type* Specify the same as **b** *type* except for footer. The default for logical page footer shall be **n** (no lines numbered).

–h *type* Specify the same as **b** *type* except for header. The default *type* for logical page header shall be **n** (no lines numbered).

108609 –**i** *incr* Specify the increment value used to number logical page lines. The default shall be
108610 1.

108611 –**l** *num* Specify the number of blank lines to be considered as one. For example, –**l 2** results
108612 in only the second adjacent blank line being numbered (if the appropriate –**h a**,
108613 –**b a**, or –**f a** option is set). The default shall be 1.

108614 –**n** *format* Specify the line numbering format. Recognized values are: **ln**, left justified, leading
108615 zeros suppressed; **rn**, right justified, leading zeros suppressed; **rz**, right justified,
108616 leading zeros kept. The default *format* shall be **rn** (right justified).

108617 –**p** Specify that numbering should not be restarted at logical page delimiters.

108618 –**s** *sep* Specify the characters used in separating the line number and the corresponding
108619 text line. The default *sep* shall be a <tab>.

108620 –**v** *startnum* Specify the initial value used to number logical page lines. The default shall be 1.

108621 –**w** *width* Specify the number of characters to be used for the line number. The default *width*
108622 shall be 6.

108623 **OPERANDS**

108624 The following operand shall be supported:

108625 *file* A pathname of a text file to be line-numbered.

108626 **STDIN**

108627 The standard input shall be used if no *file* operand is specified, and shall be used if the *file*
108628 operand is `'-'` and the implementation treats the `'-'` as meaning standard input. Otherwise,
108629 the standard input shall not be used. See the INPUT FILES section.

108630 **INPUT FILES**

108631 The input file shall be a text file.

108632 **ENVIRONMENT VARIABLES**

108633 The following environment variables shall affect the execution of *nl*:

108634 *LANG* Provide a default value for the internationalization variables that are unset or null.
108635 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
108636 variables used to determine the values of locale categories.)

108637 *LC_ALL* If set to a non-empty string value, override the values of all the other
108638 internationalization variables.

108639 *LC_COLLATE*
108640 Determine the locale for the behavior of ranges, equivalence classes, and multi-
108641 character collating elements within regular expressions.

108642 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
108643 characters (for example, single-byte as opposed to multi-byte characters in
108644 arguments and input files), the behavior of character classes within regular
108645 expressions, and for deciding which characters are in character class **graph** (for the
108646 –**b t**, –**f t**, and –**h t** options).

108647 *LC_MESSAGES*
108648 Determine the locale that should be used to affect the format and contents of
108649 diagnostic messages written to standard error.

NLSPATH Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall be a text file in the following format:

`"%s%s%s", <line number>, <separator>, <input line>`

where **<***line number***>** is one of the following numeric formats:

`%6d` When the **rn** format is used (the default; see **−n**).

`%06d` When the **rz** format is used.

`%−6d` When the **ln** format is used.

<empty> When line numbers are suppressed for a portion of the page; the **<***separator***>** is also suppressed.

In the preceding list, the number 6 is the default width; the **−w** option can change this value.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

In using the **−d** *delim* option, care should be taken to escape characters that have special meaning to the command interpreter.

EXAMPLES

The command:

```
nl −v 10 −i 10 −d \!+ file1
```

numbers *file1* starting at line number 10 with an increment of 10. The logical page delimiter is `"!+"`. Note that the `'!'` has to be escaped when using *csh* as a command interpreter because of its history substitution syntax. For *ksh* and *sh* the escape is not necessary, but does not do any harm.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pr

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 5

The option [**–f** *type*] is added to the SYNOPSIS. The option descriptions are presented in alphabetic order. The description of **–bt** is changed to ``Number only non-empty lines''.

Issue 6

The obsolescent behavior allowing the options to be intermingled with the optional *file* operand is removed.

Issue 7

Austin Group Interpretation 1003.1-2001 #092 is applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

nm — write the name list of an object file (**DEVELOPMENT**)

SYNOPSIS

SD `nm [-APv] [-g|-u] [-t format] file...`

XSI `nm [-APv] [-efox] [-g|-u] [-t format] file...`

DESCRIPTION

The *nm* utility shall display symbolic information appearing in the object file, executable file, or object-file library named by *file*. If no symbolic information is available for a valid input file, the *nm* utility shall report that fact, but not consider it an error condition.

XSI The default base used when numeric values are written is unspecified. On XSI-conformant systems, it shall be decimal if the **–P** option is not specified.

OPTIONS

The *nm* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–A Write the full pathname or library name of an object on each line.

XSI **–e** Write only external (global) and static symbol information.

XSI **–f** Produce full output. Write redundant symbols (**.text**, **.data**, and **.bss**), normally suppressed.

–g Write only external (global) symbol information.

XSI **–o** Write numeric values in octal (equivalent to **–t o**).

–P Write information in a portable output format, as specified in the STDOUT section.

–t *format* Write each numeric value in the specified format. The format shall be dependent on the single character used as the *format* option-argument:

XSI `d` decimal (default if **–P** is not specified).

`o` octal.

`x` hexadecimal (default if **–P** is specified).

–u Write only undefined symbols.

–v Sort output by value instead of by symbol name.

XSI **–x** Write numeric values in hexadecimal (equivalent to **–t x**).

OPERANDS

The following operand shall be supported:

file A pathname of an object file, executable file, or object-file library.

STDIN

See the INPUT FILES section.

INPUT FILES

The input file shall be an object file, an object-file library whose format is the same as those produced by the *ar* utility for link editing, or an executable file. The *nm* utility may accept additional implementation-defined object library formats for the input file.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *nm*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for character collation information for the symbol-name and symbol-value collation sequences.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If symbolic information is present in the input files, then for each file or for each member of an archive, the *nm* utility shall write the following information to standard output. By default, the format is unspecified, but the output shall be sorted by symbol name according to the collation sequence in the current locale.

- Library or object name, if **−A** is specified
- Symbol name
- Symbol type, which shall either be one of the following single characters or an implementation-defined type represented by a single character:

  `A` Global absolute symbol.

  `a` Local absolute symbol.

  `B` Global ``bss'' (that is, uninitialized data space) symbol.

  `b` Local bss symbol.

  `D` Global data symbol.

  `d` Local data symbol.

  `T` Global text symbol.

  `t` Local text symbol.

  `U` Undefined symbol.
- Value of the symbol
- The size associated with the symbol, if applicable

This information may be supplemented by additional information specific to the

implementation.

If the **−P** option is specified, the previous information shall be displayed using the following portable format. The three versions differ depending on whether **−t d**, **−t o**, or **−t x** was specified, respectively:

```
"%s%s %s %d %d\n", <library/object name>, <name>, <type>,
    <value>, <size>

"%s%s %s %o %o\n", <library/object name>, <name>, <type>,
    <value>, <size>

"%s%s %s %x %x\n", <library/object name>, <name>, <type>,
    <value>, <size>
```

where **<***library/object name***>** shall be formatted as follows:

- If **−A** is not specified, **<***library/object name***>** shall be an empty string.
- If **−A** is specified and the corresponding *file* operand does not name a library:

  ```
  "%s: ", <file>
  ```

- If **−A** is specified and the corresponding *file* operand names a library. In this case, **<***object file***>** shall name the object file in the library containing the symbol being described:

  ```
  "%s[%s]: ", <file>, <object file>
  ```

If **−A** is not specified, then if more than one *file* operand is specified or if only one *file* operand is specified and it names a library, *nm* shall write a line identifying the object containing the following symbols before the lines containing those symbols, in the form:

- If the corresponding *file* operand does not name a library:

  ```
  "%s:\n", <file>
  ```

- If the corresponding *file* operand names a library; in this case, **<***object file***>** shall be the name of the file in the library containing the following symbols:

  ```
  "%s[%s]:\n", <file>, <object file>
  ```

If **−P** is specified, but **−t** is not, the format shall be as if **−t x** had been specified.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Mechanisms for dynamic linking make this utility less meaningful when applied to an executable file because a dynamically linked executable may omit numerous library routines that would be found in a statically linked executable.

EXAMPLES

None.

RATIONALE

Historical implementations of *nm* have used different bases for numeric output and supplied different default types of symbols that were reported. The **–t** *format* option, similar to that used in *od* and *strings*, can be used to specify the numeric base; **–g** and **–u** can be used to restrict the amount of output or the types of symbols included in the output.

The compromise of using **–t** *format versus* using **–d**, **–o**, and other similar options was necessary because of differences in the meaning of **–o** between implementations. The **–o** option from BSD has been provided here as **–A** to avoid confusion with the **–o** from System V (which has been provided here as **–t** and as **–o** on XSI-conformant systems).

The option list was significantly reduced from that provided by historical implementations.

The *nm* description is a subset of both the System V and BSD *nm* utilities with no specified default output.

It was recognized that mechanisms for dynamic linking make this utility less meaningful when applied to an executable file (because a dynamically linked executable file may omit numerous library routines that would be found in a statically linked executable file), but the value of *nm* during software development was judged to outweigh other limitations.

The default output format of *nm* is not specified because of differences in historical implementations. The **–P** option was added to allow some type of portable output format. After a comparison of the different formats used in SunOS, BSD, SVR3, and SVR4, it was decided to create one that did not match the current format of any of these four systems. The format devised is easy to parse by humans, easy to parse in shell scripts, and does not need to vary depending on locale (because no English descriptions are included). All of the systems currently have the information available to use this format.

The format given in *nm* STDOUT uses <space> characters between the fields, which may be any number of <blank> characters required to align the columns. The single-character types were selected to match historical practice, and the requirement that implementation additions also be single characters made parsing the information easier for shell scripts.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

ar, *c17*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is marked as supported when both the User Portability Utilities option and the Software Development Utilities option are supported.

Issue 7

The *nm* utility is removed from the User Portability Utilities option. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0125 [263] and XCU/TC1-2008/0126 [263] are applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0148 [744] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1062 is applied, inserting an empty line between the two SYNOPSIS forms.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

nohup — invoke a utility immune to hangups

SYNOPSIS

```
nohup utility [argument...]
```

DESCRIPTION

The *nohup* utility shall invoke the utility named by the *utility* operand with arguments supplied as the *argument* operands. At the time the named *utility* is invoked, the SIGHUP signal shall be set to be ignored.

If standard input is associated with a terminal, the *nohup* utility may redirect standard input from an unspecified file.

If the standard output is a terminal, all output written by the named *utility* to its standard output shall be appended to the end of the file **nohup.out** in the current directory. If **nohup.out** cannot be created or opened for appending, the output shall be appended to the end of the file **nohup.out** in the directory specified by the *HOME* environment variable. If neither file can be created or opened for appending, *utility* shall not be invoked. If a file is created, the file's permission bits shall be set to S_IRUSR | S_IWUSR.

If standard error is a terminal and standard output is open but is not a terminal, all output written by the named utility to its standard error shall be redirected to the same open file description as the standard output. If standard error is a terminal and standard output either is a terminal or is closed, the same output shall instead be appended to the end of the **nohup.out** file as described above.

OPTIONS

None.

OPERANDS

The following operands shall be supported:

utility The name of a utility that is to be invoked. If the *utility* operand names any of the special built-in utilities in Section 2.15 (on page 2526), the results are undefined.

argument Any string to be supplied as an argument when invoking the utility named by the *utility* operand.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *nohup*:

HOME Determine the pathname of the user's home directory: if the output file **nohup.out** cannot be created in the current directory, the *nohup* utility shall use the directory named by *HOME* to create the file.

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

PATH Determine the search path that is used to locate the utility to be invoked. See XBD Chapter 8 (on page 167).

ASYNCHRONOUS EVENTS

The *nohup* utility shall take the standard action for all signals except that SIGHUP shall be ignored.

STDOUT

If the standard output is not a terminal, the standard output of *nohup* shall be the standard output generated by the execution of the *utility* specified by the operands. Otherwise, nothing shall be written to the standard output.

STDERR

If the standard output is a terminal, a message shall be written to the standard error, indicating the name of the file to which the output is being appended. The name of the file shall be either **nohup.out** or **$HOME/nohup.out**.

OUTPUT FILES

Output written by the named utility is appended to the file **nohup.out** (or **$HOME/nohup.out**), if the conditions hold as described in the DESCRIPTION.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

126 The utility specified by *utility* was found but could not be invoked.

127 An error occurred in the *nohup* utility or the utility specified by *utility* could not be found.

Otherwise, the exit status of *nohup* shall be that of the utility specified by the *utility* operand.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *command*, *env*, *nice*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit code 127 if a utility to be invoked cannot be found, so that applications can distinguish ``failure to find a utility'' from ``invoked utility exited with an error indication''. However, the *command* and *nohup* utilities also use exit code 127 when an error occurs in those utilities, which means exit code 127 is not universally a ``not found'' indicator. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for ``normal error conditions'' and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any

attempt to *exec* the utility fails for any other reason.

EXAMPLES

It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done by placing pipelines and command lists in a single file; this file can then be invoked as a utility, and the *nohup* applies to everything in the file.

Alternatively, the following command can be used to apply *nohup* to a complex command:

```
nohup sh -c -- 'complex-command-line' </dev/null
```

RATIONALE

The 4.3 BSD version ignores SIGTERM and SIGHUP, and if **./nohup.out** cannot be used, it fails instead of trying to use **$HOME/nohup.out**.

The *csh* utility has a built-in version of *nohup* that acts differently from the *nohup* defined in this volume of POSIX.1-2024.

The term *utility* is used, rather than *command*, to highlight the fact that shell compound commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility* includes user application programs and shell scripts, not just the standard utilities.

Historical versions of the *nohup* utility use default file creation semantics. Some more recent versions use the permissions specified here as an added security precaution.

Some historical implementations ignore SIGQUIT in addition to SIGHUP; others ignore SIGTERM. An early proposal allowed, but did not require, SIGQUIT to be ignored. Several reviewers objected that *nohup* should only modify the handling of SIGHUP as required by this volume of POSIX.1-2024.

Historical versions of *nohup* did not affect standard input, but that causes problems in the common scenario where the user logs into a system, types the command:

```
nohup make &
```

at the prompt, and then logs out. If standard input is not affected by *nohup*, the login session may not terminate for quite some time, since standard input remains open until *make* exits. To avoid this problem, POSIX.1-2024 allows implementations to redirect standard input if it is a terminal. Since the behavior is implementation-defined, portable applications that may run into the problem should redirect standard input themselves. For example, instead of:

```
nohup make &
```

an application can invoke:

```
nohup make </dev/null &
```

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 2 (on page 2472), *sh*

XBD Chapter 8 (on page 167)

XSH *signal*( )

CHANGE HISTORY

First released in Issue 2.

Issue 7

Austin Group Interpretations 1003.1-2001 #104, #105, and #106 are applied.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1530 is applied, changing "`sh -c`" to "`sh -c --`".

Austin Group Defect 1586 is applied, adding the *timeout* utility.

Austin Group Defect 1594 is applied, changing the APPLICATION USAGE section.

NAME

od — dump files in various formats

SYNOPSIS

```
od [-v] [-A address_base] [-j skip] [-N count] [-t type_string]...
    [file...]
```

XSI

```
od [-bcdosx] [file] [[+]offset[.][b]]
```

DESCRIPTION

The *od* utility shall write the contents of its input files to standard output in a user-specified format.

OPTIONS

The *od* utility shall conform to XBD Section 12.2 (on page 215), except that the order of presentation of the **–t** options and the **–bcdosx** options is significant.

The following options shall be supported:

–A *address_base*
: Specify the input offset base. See the EXTENDED DESCRIPTION section. The application shall ensure that the *address_base* option-argument is a character. The characters `'d'`, `'o'`, and `'x'` specify that the offset base shall be written in decimal, octal, or hexadecimal, respectively. The character `'n'` specifies that the offset shall not be written.

XSI **–b**
: Interpret bytes in octal. This shall be equivalent to **–t o1**.

XSI **–c**
: Interpret bytes as characters specified by the current setting of the *LC_CTYPE* category. Certain non-graphic characters appear as C escapes: `"NUL=\0"`, `"BS=\b"`, `"FF=\f"`, `"NL=\n"`, `"CR=\r"`, `"HT=\t"`; others appear as 3-digit octal numbers.

XSI **–d**
: Interpret *word*s (two-byte units) in unsigned decimal. This shall be equivalent to **–t u2**.

–j *skip*
: Jump over *skip* bytes from the beginning of the input. The *od* utility shall read or seek past the first *skip* bytes in the concatenated input files. If the combined input is not at least *skip* bytes long, the *od* utility shall write a diagnostic message to standard error and exit with a non-zero exit status.

  By default, the *skip* option-argument shall be interpreted as a decimal number. With a leading 0x or 0X, the offset shall be interpreted as a hexadecimal number; otherwise, with a leading `'0'`, the offset shall be interpreted as an octal number. Appending the character `'b'`, `'k'`, or `'m'` to offset shall cause it to be interpreted as a multiple of 512, 1 024, or 1 048 576 bytes, respectively. If the *skip* number is hexadecimal, any appended `'b'` shall be considered to be the final hexadecimal digit.

–N *count*
: Format no more than *count* bytes of input. By default, *count* shall be interpreted as a decimal number. With a leading 0x or 0X, *count* shall be interpreted as a hexadecimal number; otherwise, with a leading `'0'`, it shall be interpreted as an octal number. If *count* bytes of input (after successfully skipping, if **–j** *skip* is specified) are not available, it shall not be considered an error; the *od* utility shall format the input that is available.

XSI **–o** Interpret *word*s (two-byte units) in octal. This shall be equivalent to **–t o2**.

XSI **–s** Interpret *word*s (two-byte units) in signed decimal. This shall be equivalent to **–t d2**.

–t *type_string*

Specify one or more output types. See the EXTENDED DESCRIPTION section. The application shall ensure that the *type_string* option-argument is a string specifying the types to be used when writing the input data. The string shall consist of the type specification characters `a`, `c`, `d`, `f`, `o`, `u`, and `x`, specifying named character, character, signed decimal, floating point, octal, unsigned decimal, and hexadecimal, respectively. The type specification characters `d`, `f`, `o`, `u`, and `x` can be followed by an optional unsigned decimal integer that specifies the number of bytes to be transformed by each instance of the output type. The type specification character `f` can be followed by an optional `F`, `D`, or `L` indicating that the conversion should be applied to an item of type **float**, **double**, or **long double**, respectively. The type specification characters `d`, `o`, `u`, and `x` can be followed by an optional `C`, `S`, `I`, or `L` indicating that the conversion should be applied to an item of type **char**, **short**, **int**, or **long**, respectively. Multiple types can be concatenated within the same *type_string* and multiple **–t** options can be specified. Output lines shall be written for each type specified in the order in which the type specification characters are specified.

–v Write all input data. Without the **–v** option, any number of groups of output lines, which would be identical to the immediately preceding group of output lines (except for the byte offsets), shall be replaced with a line containing only an <asterisk> (`'*'`).

XSI **–x** Interpret *word*s (two-byte units) in hexadecimal. This shall be equivalent to **–t x2**.

XSI Multiple types can be specified by using multiple **–bcdostx** options. Output lines are written for each type specified in the order in which the types are specified.

OPERANDS

The following operands shall be supported:

file A pathname of a file to be read. If no *file* operands are specified, the standard input shall be used.

If there are no more than two operands, none of the **–A**, **–j**, **–N**, **–t**, or **–v** options is specified, and either of the following is true: the first character of the last operand is a <plus-sign> (`'+'`), or there are two operands and the first character of the last operand is numeric; XSI the last operand shall be interpreted as an offset operand on XSI-conformant systems. Under these conditions, the results are unspecified on systems that are not XSI-conformant systems.

XSI **[+]***offset***[.][b]** The *offset* operand specifies the offset in the file where dumping is to commence. This operand is normally interpreted as octal bytes. If `'.'` is appended, the offset shall be interpreted in decimal. If `'b'` is appended, the offset shall be interpreted in units of 512 bytes.

STDIN

The standard input shall be used if no *file* operands are specified, and shall be used if a *file* operand is `'–'` and the implementation treats the `'–'` as meaning standard input. Otherwise, the standard input shall not be used. See the INPUT FILES section.

INPUT FILES

The input files can be any file type.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *od*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

LC_NUMERIC
Determine the locale for selecting the radix character used when writing floating-point formatted output.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

See the EXTENDED DESCRIPTION section.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

The *od* utility shall copy sequentially each input file to standard output, transforming the input
XSI data according to the output types specified by the **–t** option or the **–bcdosx** options. If no output type is specified, the default output shall be as if **–t oS** had been specified.

The number of bytes transformed by the output type specifier `c` may be variable depending on the *LC_CTYPE* category.

The default number of bytes transformed by output type specifiers `d`, `f`, `o`, `u`, and `x` corresponds to the various C-language types as follows. If the *c17* compiler is present on the system, these specifiers shall correspond to the sizes used by default in that compiler. Otherwise, these sizes may vary among systems that conform to POSIX.1-2024.

- For the type specifier characters `d`, `o`, `u`, and `x`, the default number of bytes shall correspond to the size of the underlying implementation's basic integer type. For these specifier characters, the implementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-language types **char**, **short**, **int**, and **long**. These numbers can also be specified by an application as the characters `'C'`, `'S'`, `'I'`, and `'L'`, respectively. The implementation shall also support the values 1, 2, 4, and 8, even if it provides no C-Language types of those sizes. The

implementation shall support the decimal value corresponding to the C-language type **long long**. The byte order used when interpreting numeric values is implementation-defined, but shall correspond to the order in which a constant of the corresponding type is stored in memory on the system.

- For the type specifier character `f`, the default number of bytes shall correspond to the number of bytes in the underlying implementation's basic double precision floating-point data type. The implementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-language types **float, double**, and **long double**. These numbers can also be specified by an application as the characters `'F'`, `'D'`, and `'L'`, respectively.

The type specifier character `a` specifies that bytes shall be interpreted as named characters from the International Reference Version (IRV) of the ISO/IEC 646:1991 standard. Only the least significant seven bits of each byte shall be used for this type specification. Bytes with the values listed in the following table shall be written using the corresponding names for those characters.

Table 3-14 Named Characters in *od*

Value	Name	Value	Name	Value	Name	Value	Name
\000	**nul**	\001	**soh**	\002	**stx**	\003	**etx**
\004	**eot**	\005	**enq**	\006	**ack**	\007	**bel**
\010	**bs**	\011	**ht**	\012	**lf** or **nl***	\013	**vt**
\014	**ff**	\015	**cr**	\016	**so**	\017	**si**
\020	**dle**	\021	**dc1**	\022	**dc2**	\023	**dc3**
\024	**dc4**	\025	**nak**	\026	**syn**	\027	**etb**
\030	**can**	\031	**em**	\032	**sub**	\033	**esc**
\034	**fs**	\035	**gs**	\036	**rs**	\037	**us**
\040	**sp**	\177	**del**				

Note: The `"\012"` value may be written either as **lf** or **nl**.

The type specifier character `c` specifies that bytes shall be interpreted as characters specified by the current setting of the *LC_CTYPE* locale category. Characters listed in the table in XBD Chapter 5 (on page 113) (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`) shall be written as the corresponding escape sequences, except that <backslash> shall be written as a single <backslash> and a NUL shall be written as `'\0'`. Other non-printable characters shall be written as one three-digit octal number for each byte in the character. Printable multi-byte characters shall be written in the area corresponding to the first byte of the character; the two-character sequence `"**"` shall be written in the area corresponding to each remaining byte in the character, as an indication that the character is continued. When either the **–j** *skip* or **–N** *count* option is specified along with the `c` type specifier, and this results in an attempt to start or finish in the middle of a multi-byte character, the result is implementation-defined.

The input data shall be manipulated in blocks, where a block is defined as a multiple of the least common multiple of the number of bytes transformed by the specified output types. If the least common multiple is greater than 16, the results are unspecified. Each input block shall be written as transformed by each output type, one per written line, in the order that the output types were specified. If the input block size is larger than the number of bytes transformed by the output type, the output type shall sequentially transform the parts of the input block, and the output from each of the transformations shall be separated by one or more <blank> characters.

If, as a result of the specification of the **–N** option or end-of-file being reached on the last input file, input data only partially satisfies an output type, the input shall be extended sufficiently

with null bytes to write the last byte of the input.

Unless −**A n** is specified, the first output line produced for each input block shall be preceded by the input offset, cumulative across input files, of the next byte to be written. The format of the input offset is unspecified; however, it shall not contain any <blank> characters, shall start at the first character of the output line, and shall be followed by one or more <blank> characters. In addition, the offset of the byte following the last byte written shall be written after all the input data has been processed, but shall not be followed by any <blank> characters. If −**A n** is specified, it is unspecified whether the line that would contain this final offset is written as an empty line or is not written.

If no −**A** option is specified, the input offset base is unspecified.

EXIT STATUS

The following exit values shall be returned:

0 All input files were processed successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

XSI-conformant applications are warned not to use filenames starting with '+' or a first operand starting with a numeric character so that the old functionality can be maintained by implementations, unless they specify one of the −**A**, −**j**, or −**N** options. To guarantee that one of these filenames is always interpreted as a filename, an application could always specify the address base format with the −**A** option.

EXAMPLES

If a file containing 128 bytes with decimal values zero to 127, in increasing order, is supplied as standard input to the command:

```
od -A d -t a
```

on an implementation using an input block size of 16 bytes, the standard output, independent of the current locale setting, would be similar to:

```
0000000 nul soh stx etx eot enq ack bel  bs  ht  nl  vt  ff  cr  so  si
0000016 dle dc1 dc2 dc3 dc4 nak syn etb can  em sub esc  fs  gs  rs  us
0000032  sp   !   "   #   $   %   &   '   (   )   *   +   ,   -   .   /
0000048   0   1   2   3   4   5   6   7   8   9   :   ;   <   =   >   ?
0000064   @   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O
0000080   P   Q   R   S   T   U   V   W   X   Y   Z   [   \   ]   ^   _
0000096   `   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o
0000112   p   q   r   s   t   u   v   w   x   y   z   {   |   }   ~ del
0000128
```

Note that this volume of POSIX.1-2024 allows **nl** or **lf** to be used as the name for the ISO/IEC 646:1991 standard IRV character with decimal value 10. The IRV names this character **lf** (line feed), but traditional implementations have referred to this character as newline (**nl**) and the POSIX locale character set symbolic name for the corresponding character is a <newline>.

The command:

```
od -A o -t o2x2x -N 18
```

on a system with 32-bit words and an implementation using an input block size of 16 bytes

could write 18 bytes in approximately the following format:

```
0000000 032056 031440 041123 042040 052516 044530 020043 031464
          342e   3320   4253   4420   554e   4958   2023   3334
             342e3320       42534420       554e4958       20233334
0000020 032472
          353a
             353a0000
0000022
```

The command:

```
od -A d -t f -t o4 -t x4 -N 24 -j 0x15
```

on a system with 64-bit doubles (for example, IEEE Std 754-1985 double precision floating-point format) would skip 21 bytes of input data and then write 24 bytes in approximately the following format:

```
0000000    1.00000000000000e+00    1.57350000000000e+01
         07774000000 00000000000 10013674121 35341217270
            3ff00000    00000000    402f3851    eb851eb8
0000016    1.40668230000000e+02
         10030312542 04370303230
            40619562    23e18698
0000024
```

RATIONALE

The *od* utility went through several names in early proposals, including *hd*, *xd*, and most recently *hexdump*. There were several objections to all of these based on the following reasons:

- The *hd* and *xd* names conflicted with historical utilities that behaved differently.
- The *hexdump* description was much more complex than needed for a simple dump utility.
- The *od* utility has been available on all historical implementations and there was no need to create a new name for a utility so similar to the historical *od* utility.

The original reasons for not standardizing historical *od* were also fairly widespread. Those reasons are given below along with rationale explaining why the standard developers believe that this version does not suffer from the indicated problem:

- The BSD and System V versions of *od* have diverged, and the intersection of features provided by both does not meet the needs of the user community. In fact, the System V version only provides a mechanism for dumping octal bytes and **short**s, signed and unsigned decimal **short**s, hexadecimal **short**s, and ASCII characters. BSD added the ability to dump **float**s, **double**s, named ASCII characters, and octal, signed decimal, unsigned decimal, and hexadecimal **long**s. The version presented here provides more normalized forms for dumping bytes, **short**s, **int**s, and **long**s in octal, signed decimal, unsigned decimal, and hexadecimal; **float**, **double**, and **long double**; and named ASCII as well as current locale characters.
- It would not be possible to come up with a compatible superset of the BSD and System V flags that met the requirements of the standard developers. The historical default *od* output is the specified default output of this utility. None of the option letters chosen for this version of *od* conflict with any of the options to historical versions of *od*.

- On systems with different sizes for **short**, **int**, and **long**, there was no way to ask for dumps of **int**s, even in the BSD version. Because of the way options are named, the name space could not be extended to solve these problems. This is why the **−t** option was added (with type specifiers more closely matched to the *printf*( ) formats used in the rest of this volume of POSIX.1-2024) and the optional field sizes were added to the `d`, `f`, `o`, `u`, and `x` type specifiers. It is also one of the reasons why the historical practice was not mandated as a required obsolescent form of *od*. (Although the old versions of *od* are not listed as an obsolescent form, implementations are urged to continue to recognize the older forms for several more years.) The `a`, `c`, `f`, `o`, and `x` types match the meaning of the corresponding format characters in the historical implementations of *od* except for the default sizes of the fields converted. The `d` format is signed in this volume of POSIX.1-2024 to match the *printf*( ) notation. (Historical versions of *od* used `d` as a synonym for `u` in this version. The System V implementation uses `s` for signed decimal; BSD uses `i` for signed decimal and `s` for null-terminated strings.) Other than `d` and `u`, all of the type specifiers match format characters in the historical BSD version of **od**.

  The sizes of the C-language types **char**, **short**, **int**, **long**, **float**, **double**, and **long double** are used even though it is recognized that there may be zero or more than one compiler for the C language on an implementation and that they may use different sizes for some of these types. (For example, one compiler might use 2 bytes **short**s, 2 bytes **int**s, and 4 bytes **long**s, while another compiler (or an option to the same compiler) uses 2 bytes **short**s, 4 bytes **int**s, and 4 bytes **long**s.) Nonetheless, there has to be a basic size known by the implementation for these types, corresponding to the values reported by invocations of the *getconf* utility when called with *system_var* operands {UCHAR_MAX}, {USHORT_MAX}, {UINT_MAX}, and {ULONG_MAX} for the types **char**, **short**, **int**, and **long**, respectively. There are similar constants required by the ISO C standard, but not required by the System Interfaces volume of POSIX.1-2024 or this volume of POSIX.1-2024. They are {FLT_MANT_DIG}, {DBL_MANT_DIG}, and {LDBL_MANT_DIG} for the types **float**, **double**, and **long double**, respectively. If the optional *c17* utility is provided by the implementation and used as specified by this volume of POSIX.1-2024, these are the sizes that would be provided. If an option is used that specifies different sizes for these types, there is no guarantee that the *od* utility is able to interpret binary data output by such a program correctly.

  This volume of POSIX.1-2024 requires that the numeric values of these lengths be recognized by the *od* utility and that symbolic forms also be recognized. Thus, a conforming application can always look at an array of **unsigned long** data elements using *od* **−t** *uL*.

- The method of specifying the format for the address field based on specifying a starting offset in a file unnecessarily tied the two together. The **−A** option now specifies the address base and the **−S** option specifies a starting offset.

- It would be difficult to break the dependence on US ASCII to achieve an internationalized utility. It does not seem to be any harder for *od* to dump characters in the current locale than it is for the *ed* or *sed* **l** commands. The `c` type specifier does this without difficulty and is completely compatible with the historical implementations of the **c** format character when the current locale uses a superset of the ISO/IEC 646:1991 standard as a codeset. The `a` type specifier (from the BSD **a** format character) was left as a portable means to dump ASCII (or more correctly ISO/IEC 646:1991 standard (IRV)) so that headers produced by *pax* could be deciphered even on systems that do not use the ISO/IEC 646:1991 standard as a subset of their base codeset.

The use of `"**"` as an indication of continuation of a multi-byte character in `c` specifier output

was chosen based on seeing an implementation that uses this method. The continuation bytes have to be marked in a way that is not ambiguous with another single-byte or multi-byte character.

An early proposal used **–S** and **–n**, respectively, for the **–j** and **–N** options eventually selected. These were changed to avoid conflicts with historical implementations.

The original standard specified **–t o2** as the default when no output type was given. This was changed to **–t oS** (the length of a **short**) to accommodate a supercomputer implementation that historically used 64 bits as its default (and that defined shorts as 64 bits). This change should not affect conforming applications. The requirement to support lengths of 1, 2, and 4 was added at the same time to address an historical implementation that had no two-byte data types in its C compiler.

The use of a basic integer data type is intended to allow the implementation to choose a word size commonly used by applications on that architecture.

Earlier versions of this standard allowed for implementations with bytes other than eight bits, but this has been modified in this version.

FUTURE DIRECTIONS

All option and operand interfaces marked XSI may be removed in a future version.

SEE ALSO

c17, *sed*

XBD Chapter 5 (on page 113), Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 5

In the description of the **–c** option, the phrase ``This is equivalent to **–t c**.'' is deleted.

The FUTURE DIRECTIONS section is modified.

Issue 6

The *od* utility is changed to remove the assumption that **short** was a two-byte entity, as per the revisions in the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/33 is applied, correcting the examples which used an undefined **–n** option, which should have been **–N**.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/19 is applied, removing text describing behavior on systems with bytes consisting of more than eight bits.

Issue 7

Austin Group Interpretation 1003.1-2001 #092 is applied.

SD5-XCU-ERN-37 is applied, updating the OPERANDS section.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 1017 is applied, clarifying that when **–A n** is specified, the line that would contain the final offset can either be written as an empty line or not be written.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

paste — merge corresponding or subsequent lines of files

SYNOPSIS

```
paste [-s] [-d list] file...
```

DESCRIPTION

The *paste* utility shall concatenate the corresponding lines of the given input files, and write the resulting lines to standard output.

The default operation of *paste* shall concatenate the corresponding lines of the input files. The <newline> of every line except the line from the last input file shall be replaced with a <tab>.

If an end-of-file condition is detected on one or more input files, but not all input files, *paste* shall behave as though empty lines were read from the files on which end-of-file was detected, unless the –**s** option is specified.

OPTIONS

The *paste* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–**d** *list* Unless a <backslash> character appears in *list*, each character in *list* is an element specifying a delimiter character. If a <backslash> character appears in *list*, the <backslash> character and one or more characters following it are an element specifying a delimiter character as described below. These elements specify one or more delimiters to use, instead of the default <tab>, to replace the <newline> of the input lines. The elements in *list* shall be used circularly; that is, when the list is exhausted the first element from the list is reused. When the –**s** option is specified:

- The last <newline> in a file shall not be modified.
- The delimiter shall be reset to the first element of *list* after each *file* operand is processed.

When the –**s** option is not specified:

- The <newline> characters in the file specified by the last *file* operand shall not be modified.
- The delimiter shall be reset to the first element of list each time a line is processed from each file.

If a <backslash> character appears in *list*, it and the character following it shall be used to represent the following delimiter characters:

`\n` <newline>.

`\t` <tab>.

`\\` <backslash> character.

`\0` Empty string (not a null character). If `'\0'` is immediately followed by the character `'x'`, the character `'X'`, or any character defined by the *LC_CTYPE* **digit** keyword (see XBD Chapter 7, on page 127), the results are unspecified.

If any other characters follow the <backslash>, the results are unspecified.

–**s** Concatenate all of the lines from each input file into one line of output per file, in command line order. The <newline> of every line except the last line in each input file shall be replaced with a <tab>, unless otherwise specified by the –**d** option. If an input file is empty, the output line corresponding to that file shall consist of

109415 only a <newline> character.

109416 **OPERANDS**

109417 The following operand shall be supported:

109418 *file* A pathname of an input file. If `'-'` is specified for one or more of the *file*s, the
109419 standard input shall be used; the standard input shall be read one line at a time,
109420 circularly, for each instance of `'-'`. Implementations shall support pasting of at
109421 least 12 *file* operands.

109422 **STDIN**

109423 The standard input shall be used only if one or more *file* operands is `'-'`. See the INPUT FILES
109424 section.

109425 **INPUT FILES**

109426 The input files shall be text files, except that line lengths shall be unlimited.

109427 **ENVIRONMENT VARIABLES**

109428 The following environment variables shall affect the execution of *paste*:

109429 *LANG* Provide a default value for the internationalization variables that are unset or null.
109430 (See XBD Section 8.2 (on page 169) the precedence of internationalization variables
109431 used to determine the values of locale categories.)

109432 *LC_ALL* If set to a non-empty string value, override the values of all the other
109433 internationalization variables.

109434 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
109435 characters (for example, single-byte as opposed to multi-byte characters in
109436 arguments and input files).

109437 *LC_MESSAGES*

109438 Determine the locale that should be used to affect the format and contents of
109439 diagnostic messages written to standard error.

109440 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

109441 **ASYNCHRONOUS EVENTS**

109442 Default.

109443 **STDOUT**

109444 Concatenated lines of input files shall be separated by the <tab> (or other characters under the
109445 control of the **–d** option) and terminated by a <newline>.

109446 **STDERR**

109447 The standard error shall be used only for diagnostic messages.

109448 **OUTPUT FILES**

109449 None.

109450 **EXTENDED DESCRIPTION**

109451 None.

109452 **EXIT STATUS**

109453 The following exit values shall be returned:

109454 0 Successful completion.

109455 >0 An error occurred.

CONSEQUENCES OF ERRORS

If one or more input files cannot be opened when the **–s** option is not specified, a diagnostic message shall be written to standard error, but no output is written to standard output. If the **–s** option is specified, the *paste* utility shall provide the default behavior described in Section 1.4 (on page 2462).

APPLICATION USAGE

When the escape sequences of the *list* option-argument are used in a shell script, they must be quoted; otherwise, the shell treats the <backslash> as a special character.

Conforming applications should only use the specific <backslash>-escaped delimiters presented in this volume of POSIX.1-2024. Historical implementations treat `'\x'`, where `'x'` is not in this list, as `'x'`, but future implementations are free to expand this list to recognize other common escapes similar to those accepted by *printf* and other standard utilities.

Most of the standard utilities work on text files. The *cut* utility can be used to turn files with arbitrary line lengths into a set of text files containing the same data. The *paste* utility can be used to create (or recreate) files with arbitrary line lengths. For example, if *file* contains long lines:

```
cut -b 1-500 -n file > file1
cut -b 501- -n file > file2
```

creates **file1** (a text file) with lines no longer than 500 bytes (plus the <newline>) and **file2** that contains the remainder of the data from *file*. Note that **file2** is not a text file if there are lines in *file* that are longer than 500 + {LINE_MAX} bytes. The original file can be recreated from **file1** and **file2** using the command:

```
paste -d "\0" file1 file2 > file
```

The commands:

```
paste -d "\0" ...
paste -d "" ...
```

are not necessarily equivalent; the latter is not specified by this volume of POSIX.1-2024 and may result in an error. The construct `'\0'` is used to mean ``no separator'' because historical versions of *paste* did not follow the syntax guidelines, and the command:

```
paste -d"" ...
```

could not be handled properly by *getopt*( ).

EXAMPLES

1. Write out a directory in four columns:

   ```
   ls | paste - - - -
   ```

2. Combine pairs of lines from a file into single lines:

   ```
   paste -s -d "\t\n" file
   ```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 1.4 (on page 2462), *cut*, *grep*, *pr*

XBD Chapter 7 (on page 127), Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0149 [973] is applied.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

patch — apply changes to files

SYNOPSIS

```
patch [-blNR] [-c|-e|-n|-u] [-d dir] [-D define] [-i patchfile]
    [-o outfile] [-p num] [-r rejectfile] [file]
```

DESCRIPTION

The *patch* utility shall read a source (patch) file containing any of four forms of difference (diff) listings produced by the *diff* utility (normal, copied context, unified context, or in the style of *ed*) and apply those differences to a file. By default, *patch* shall read from the standard input.

The *patch* utility shall attempt to determine the type of the *diff* listing, unless overruled by a **−c**, **−e**, **−n**, or **−u** option.

If the patch file contains more than one patch, *patch* shall attempt to apply each of them as if they came from separate patch files. (In this case, the application shall ensure that the name of the patch file is determinable for each *diff* listing.)

OPTIONS

The *patch* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−b Save a copy of the original contents of each modified file, before the differences are applied, in a file of the same name with the suffix **.orig** appended to it. If the file already exists, it shall be overwritten; if multiple patches are applied to the same file, the **.orig** file shall be written only for the first patch. When the **−o** *outfile* option is also specified, *file***.orig** shall not be created but, if *outfile* already exists, *outfile***.orig** shall be created.

−c Interpret the patch file as a copied context difference (the output of the utility *diff* when the **−c** or **−C** options are specified).

−d *dir* Change the current directory to *dir* before processing as described in the EXTENDED DESCRIPTION section.

−D *define* Mark changes with one of the following C preprocessor constructs:

```
#ifdef define
...
#endif
```

```
#ifndef define
...
#endif
```

optionally combined with the C preprocessor construct **#else**. If the patched file is processed with the C preprocessor, where the macro *define* is defined, the output shall contain the changes from the patch file; otherwise, the output shall not contain the patches specified in the patch file.

−e Interpret the patch file as an *ed* script, rather than a *diff* script.

−i *patchfile* Read the patch information from the file named by the pathname *patchfile*, rather than the standard input.

−l (The letter ell.) Cause any sequence of <blank> characters in the difference script to match any sequence of <blank> characters in the input file. Other characters shall be matched exactly.

109551 **−n** Interpret the script as a normal difference.

109552 **−N** Ignore patches where the differences have already been applied to the file; by
109553 default, already-applied patches shall be rejected.

109554 **−o** *outfile* Instead of modifying the files (specified by the *file* operand or the difference
109555 listings) directly, write a copy of the file referenced by each patch, with the
109556 appropriate differences applied, to *outfile*. Multiple patches for a single file shall be
109557 applied to the intermediate versions of the file created by any previous patches,
109558 and shall result in multiple, concatenated versions of the file being written to
109559 *outfile*.

109560 **−p** *num* For all pathnames in the patch file that indicate the names of files to be patched,
109561 delete *num* pathname components from the beginning of each pathname. If the
109562 pathname in the patch file is absolute, any leading <slash> characters shall be
109563 considered the first component (that is, **−p 1** shall remove the leading <slash>
109564 characters). Specifying **−p 0** shall cause the full pathname to be used. If **−p** is not
109565 specified, only the basename (the final pathname component) shall be used.

109566 **−R** Reverse the sense of the patch script; that is, assume that the difference script was
109567 created from the new version to the old version. The **−R** option cannot be used
109568 with *ed* scripts. The *patch* utility shall attempt to reverse each portion of the script
109569 before applying it. Rejected differences shall be saved in swapped format. If this
109570 option is not specified, and until a portion of the patch file is successfully applied,
109571 *patch* attempts to apply each portion in its reversed sense as well as in its normal
109572 sense. If the attempt is successful, the user shall be prompted to determine whether
109573 the **−R** option should be set.

109574 **−r** *rejectfile* Override the default reject filename. In the default case, the reject file shall have the
109575 same name as the output file, with the suffix **.rej** appended to it; see Patch
109576 Application (on page 3241).

109577 **−u** Interpret the patch file as a unified context difference (the output of the *diff* utility
109578 when the **−u** or **−U** options are specified).

109579 **OPERANDS**

109580 The following operand shall be supported:

109581 *file* A pathname of a file to patch.

109582 **STDIN**

109583 See the INPUT FILES section.

109584 **INPUT FILES**

109585 Input files shall be text files.

109586 **ENVIRONMENT VARIABLES**

109587 The following environment variables shall affect the execution of *patch*:

109588 *LANG* Provide a default value for the internationalization variables that are unset or null.
109589 (See XBD Section 8.2 (on page 169) the precedence of internationalization variables
109590 used to determine the values of locale categories.)

109591 *LC_ALL* If set to a non-empty string value, override the values of all the other
109592 internationalization variables.

109593 *LC_COLLATE*
109594 Determine the locale for the behavior of ranges, equivalence classes, and multi-
109595 character collating elements used in the extended regular expression defined for

the **yesexpr** locale keyword in the *LC_MESSAGES* category.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), and the behavior of character classes used in the extended regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES* category.

LC_MESSAGES
Determine the locale used to process affirmative responses, and the locale used to affect the format and contents of diagnostic messages and prompts written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

LC_TIME Determine the locale for recognizing the format of file timestamps written by the *diff* utility in a context-difference input file.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used for diagnostic and informational messages.

OUTPUT FILES

The output of the *patch* utility, the save files (**.orig** suffixes), and the reject files (**.rej** suffixes) shall be text files.

EXTENDED DESCRIPTION

A patch file may contain patching instructions for more than one file; filenames shall be determined as specified in Filename Determination (on page 3241). When the **−b** option is specified, for each patched file, the original shall be saved in a file of the same name with the suffix **.orig** appended to it.

For each patched file, a reject file may also be created as noted in Patch Application (on page 3241). In the absence of a **−r** option, the name of this file shall be formed by appending the suffix **.rej** to the original filename.

Patch File Format

The patch file shall contain zero or more lines of header information followed by one or more patches. Each patch shall contain zero or more lines of filename identification in the format produced by the **−c**, **−C**, **−u**, or **−U** options of the *diff* utility, and one or more sets of *diff* output, which are customarily called *hunks*.

The *patch* utility shall recognize the following expression in the header information:

Index: *pathname*
The file to be patched is named *pathname*.

If all lines (including headers) within a patch begin with the same leading sequence of <blank> characters, the *patch* utility shall remove this sequence before proceeding. Within each patch, if the type of difference is common context, the *patch* utility shall recognize the following expressions:

109638 *** *filename timestamp*
109639 The patches arose from *filename*.

109640 −−− *filename timestamp*
109641 The patches should be applied to *filename*.

109642 If the type of difference is unified context, the *patch* utility shall recognize the following
109643 expressions:

109644 −−− *filename timestamp*
109645 The patches arose from *filename*.

109646 +++ *filename timestamp*
109647 The patches should be applied to *filename*.

109648 Each hunk within a patch shall be the *diff* output to change a line range within the original file.
109649 The line numbers for successive hunks within a patch shall occur in ascending order.

109650 **Filename Determination**

109651 If no *file* operand is specified, *patch* shall perform the following steps to determine the filename
109652 to use:

109653 1. If the type of *diff* is context, the *patch* utility shall delete pathname components (as
109654 specified by the −**p** option) from the filename on the line beginning with `"***"` (if copied
109655 context) or `"---"` (if unified context), then test for the existence of this file relative to the
109656 current directory (or the directory specified with the −**d** option). If the file exists, the *patch*
109657 utility shall use this filename.

109658 2. If the type of *diff* is context, the *patch* utility shall delete the pathname components (as
109659 specified by the −**p** option) from the filename on the line beginning with `"---"` (if copied
109660 context) or `"+++"` (if unified context), then test for the existence of this file relative to the
109661 current directory (or the directory specified with the −**d** option). If the file exists, the *patch*
109662 utility shall use this filename.

109663 3. If the header information contains a line beginning with the string **Index:**, the *patch* utility
109664 shall delete pathname components (as specified by the −**p** option) from this line, then test
109665 for the existence of this file relative to the current directory (or the directory specified
109666 with the −**d** option). If the file exists, the *patch* utility shall use this filename.

109667 XSI 4. If an **SCCS** directory exists in the current directory, *patch* shall attempt to perform a *get* −**e**
109668 **SCCS/s.***filename* command to retrieve an editable version of the file. If the file exists, the
109669 *patch* utility shall use this filename.

109670 5. The *patch* utility shall write a prompt to standard output and request a filename
109671 interactively from the controlling terminal (for example, **/dev/tty**).

109672 **Patch Application**

109673 If the −**c**, −**e**, −**n**, or −**u** option is present, the *patch* utility shall interpret information within each
109674 hunk as a copied context difference, an *ed* difference, a normal difference, or a unified context
109675 difference, respectively. In the absence of any of these options, the *patch* utility shall determine
109676 the type of difference based on the format of information within the hunk.

109677 For each hunk, the *patch* utility shall begin to search for the place to apply the patch at the line
109678 number at the beginning of the hunk, plus or minus any offset used in applying the previous
109679 hunk. If lines matching the hunk context are not found, *patch* shall scan both forwards and
109680 backwards at least 1 000 bytes for a set of lines that match the hunk context.

If no such place is found and it is a context difference, then another scan shall take place, ignoring the first and last line of context. If that fails, the first two and last two lines of context shall be ignored and another scan shall be made. Implementations may search more extensively for installation locations.

If no location can be found, the *patch* utility shall append the hunk to the reject file. A rejected hunk that is a copied context difference, an *ed* difference, or a normal difference shall be written in copied-context-difference format regardless of the format of the patch file. It is implementation-defined whether a rejected hunk that is a unified context difference is written in copied-context-difference format or in unified-context-difference format. If the input was a normal or *ed*-style difference, the reject file may contain differences with zero lines of context. The line numbers on the hunks in the reject file may be different from the line numbers in the patch file since they shall reflect the approximate locations for the failed hunks in the new file rather than the old one.

If the type of patch is an *ed* diff, the implementation may accomplish the patching by invoking the *ed* utility.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

1 One or more lines were written to a reject file.

>1 An error occurred.

CONSEQUENCES OF ERRORS

Patches that cannot be correctly placed in the file shall be written to a reject file.

APPLICATION USAGE

The **−R** option does not work with *ed* scripts because there is too little information to reconstruct the reverse operation.

The **−p** option makes it possible to customize a patch file to local user directory structures without manually editing the patch file. For example, if the filename in the patch file was:

```
/curds/whey/src/blurfl/blurfl.c
```

Setting **−p 0** gives the entire pathname unmodified; **−p 1** gives:

```
curds/whey/src/blurfl/blurfl.c
```

without the leading <slash>, **−p 4** gives:

```
blurfl/blurfl.c
```

and not specifying **−p** at all gives:

```
blurfl.c .
```

EXAMPLES

None.

RATIONALE

Some of the functionality in historical *patch* implementations was not specified. The following documents those features present in historical implementations that have not been specified.

A deleted piece of functionality was the `'+'` pseudo-option allowing an additional set of options and a patch file operand to be given. This was seen as being insufficiently useful to standardize.

In historical implementations, if the string `"Prereq:"` appeared in the header, the *patch* utility

would search for the corresponding version information (the string specified in the header, delimited by <blank> characters or the beginning or end of a line or the file) anywhere in the original file. This was deleted as too simplistic and insufficiently trustworthy a mechanism to standardize. For example, if:

```
Prereq: 1.2
```

were in the header, the presence of a delimited 1.2 anywhere in the file would satisfy the prerequisite.

The following options were dropped from historical implementations of *patch* as insufficiently useful to standardize:

−b The **−b** option historically provided a method for changing the name extension of the backup file from the default **.orig**. This option has been modified and retained in this volume of POSIX.1-2024.

−F The **−F** option specified the number of lines of a context diff to ignore when searching for a place to install a patch.

−f The **−f** option historically caused *patch* not to request additional information from the user.

−r The **−r** option historically provided a method of overriding the extension of the reject file from the default **.rej**.

−s The **−s** option historically caused *patch* to work silently unless an error occurred.

−x The **−x** option historically set internal debugging flags.

In some file system implementations, the saving of a **.orig** file may produce unwanted results. In the case of 12, 13, or 14-character filenames (on file systems supporting 14-character maximum filenames), the **.orig** file overwrites the new file. The reject file may also exceed this filename limit. It was suggested, due to some historical practice, that a <tilde> (`'~'`) suffix be used instead of **.orig** and some other character instead of the **.rej** suffix. This was rejected because it is not obvious to the user which file is which. The suffixes **.orig** and **.rej** are clearer and more understandable.

The **−b** option has the opposite sense in some historical implementations—do not save the **.orig** file. The default case here is not to save the files, making *patch* behave more consistently with the other standard utilities.

The **−w** option in early proposals was changed to **−l** to match historical practice.

The **−N** option was included because without it, a non-interactive application cannot reject previously applied patches. For example, if a user is piping the output of *diff* into the *patch* utility, and the user only wants to patch a file to a newer version non-interactively, the **−N** option is required.

Changes to the **−l** option description were proposed to allow matching across <newline> characters in addition to just <blank> characters. Since this is not historical practice, and since some ambiguities could result, it is suggested that future developments in this area utilize another option letter, such as **−L**.

The **−u** option of GNU *patch* has been added, along with support for unified context formats.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

diff, *ed*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The description of the –**D** option and the steps in Filename Determination (on page 3241) are changed to match historical practice as defined in the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/34 is applied, clarifying the way that the *patch* utility performs **ifdef** selection for the –**D** option.

Issue 7

The *patch* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

SD5-XCU-ERN-103 and SD5-XCU-ERN-120 are applied, adding the –**u** option.

Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the *LC_MESSAGES* and *LC_CTYPE* environment variables and adding the *LC_COLLATE* environment variable.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

pathchk — check pathnames

SYNOPSIS

```
pathchk [-p] [-P] pathname...
```

DESCRIPTION

The *pathchk* utility shall check that one or more pathnames are valid (that is, they could be used to access or create a file without causing syntax errors) and portable (that is, no filename truncation results). More extensive portability checks are provided by the **–p** and **–P** options.

By default, the *pathchk* utility shall check each component of each *pathname* operand based on the underlying file system. A diagnostic shall be written for each *pathname* operand that:

- Is longer than {PATH_MAX} bytes (see **Pathname Variable Values** in XBD **<limits.h>**)
- Contains any component longer than {NAME_MAX} bytes in its containing directory
- Contains any component in a directory that is not searchable
- Contains any byte sequence that is not valid in its containing directory

The format of the diagnostic message is not specified, but shall indicate the error detected and the corresponding *pathname* operand.

It shall not be considered an error if one or more components of a *pathname* operand do not exist as long as a file matching the pathname specified by the missing components could be created that does not violate any of the checks specified above.

OPTIONS

The *pathchk* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

–p Instead of performing checks based on the underlying file system, write a diagnostic for each *pathname* operand that:

- Is longer than {_POSIX_PATH_MAX} bytes (see **Minimum Values** in XBD **<limits.h>**)
- Contains any component longer than {_POSIX_NAME_MAX} bytes
- Contains any character in any component that is not in the portable filename character set

–P Write a diagnostic for each *pathname* operand that:

- Contains a component whose first character is the <hyphen-minus> character
- Is empty

OPERANDS

The following operand shall be supported:

pathname A pathname to be checked.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *pathchk*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All *pathname* operands passed all of the checks.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *test* utility can be used to determine whether a given pathname names an existing file; it does not, however, give any indication of whether or not any component of the pathname was truncated in a directory where the _POSIX_NO_TRUNC feature is not in effect. The *pathchk* utility does not check for file existence; it performs checks to determine whether a pathname does exist or could be created with no pathname component truncation.

The *noclobber* option in the shell (see the *set* special built-in) can be used to atomically create a file. As with all file creation semantics in the System Interfaces volume of POSIX.1-2024, it guarantees atomic creation, but still depends on applications to agree on conventions and cooperate on the use of files after they have been created.

To verify that a pathname meets the requirements of filename portability, applications should

use both the **–p** and **–P** options together.

EXAMPLES

To verify that all pathnames in an imported data interchange archive are legitimate and unambiguous on the current system:

```
# This example assumes that no pathnames in the archive
# contain <newline> characters.
pax -f archive | sed -e 's/[^[:alnum:]]/\\&/g' | xargs pathchk --
if [ $? -eq 0 ]
then
    pax -r -f archive
else
    echo Investigate problems before importing files.
    exit 1
fi
```

To verify that all files in the current directory hierarchy could be moved to any system conforming to the System Interfaces volume of POSIX.1-2024 that also supports the *pax* utility:

```
find . -exec pathchk -p -P {} +
if [ $? -eq 0 ]
then
    pax -w -f ../archive .
else
    echo Portable archive cannot be created.
    exit 1
fi
```

To verify that a user-supplied pathname names a readable file and that the application can create a file extending the given path without truncation and without overwriting any existing file:

```
case $- in
    *C*)    reset="";;
    *)      reset="set +C"
            set -C;;
esac
test -r "$path" && pathchk "$path.out" &&
    rm "$path.out" > "$path.out"
if [ $? -ne 0 ]; then
    printf "%s: %s not found or %s.out fails \
creation checks.\n" $0 "$path" "$path"
    $reset    # Reset the noclobber option in case a trap
              # on EXIT depends on it.
    exit 1
fi
$reset
PROCESSING < "$path" > "$path.out"
```

The following assumptions are made in this example:

1. **PROCESSING** represents the code that is used by the application to use **$path** once it is verified that **$path.out** works as intended.

2. The state of the *noclobber* option is unknown when this code is invoked and should be set on exit to the state it was in when this code was invoked. (The **reset** variable is used in this example to restore the initial state.)

3. Note the usage of:

```
rm "$path.out" > "$path.out"
```

   a. The *pathchk* command has already verified, at this point, that **$path.out** is not truncated.

   b. With the *noclobber* option set, the shell verifies that **$path.out** does not already exist before invoking *rm*.

   c. If the shell succeeded in creating **$path.out**, *rm* removes it so that the application can create the file again in the **PROCESSING** step.

   d. If the **PROCESSING** step wants the file to exist already when it is invoked, the:

```
rm "$path.out" > "$path.out"
```

   should be replaced with:

```
> "$path.out"
```

   which verifies that the file did not already exist, but leaves **$path.out** in place for use by **PROCESSING**.

RATIONALE

The *pathchk* utility was new for the ISO POSIX-2: 1993 standard. It, along with the *set* **−C**(*noclobber*) option added to the shell, replaces the *mktemp*, *validfnam*, and *create* utilities that appeared in early proposals. All of these utilities were attempts to solve several common problems:

- Verify the validity (for several different definitions of ``valid'') of a pathname supplied by a user, generated by an application, or imported from an external source.
- Atomically create a file.
- Perform various string handling functions to generate a temporary filename.

The *create* utility, included in an early proposal, provided checking and atomic creation in a single invocation of the utility; these are orthogonal issues and need not be grouped into a single utility. Note that the *noclobber* option also provides a way of creating a lock for process synchronization; since it provides an atomic *create*, there is no race between a test for existence and the following creation if it did not exist.

Having a function like *tmpnam*( ) in the ISO C standard is important in many high-level languages. The shell programming language, however, has built-in string manipulation facilities, making it very easy to construct temporary filenames. The names needed obviously depend on the application, but are frequently of a form similar to:

$TMPDIR/***application_abbreviation***$$.*suffix*

In cases where there is likely to be contention for a given suffix, a simple shell **for** or **while** loop can be used with the shell *noclobber* option to create a file without risk of collisions, as long as applications trying to use the same filename name space are cooperating on the use of files after they have been created.

For historical purposes, **−p** does not check for the use of the <hyphen-minus> character as the first character in a component of the pathname, or for an empty *pathname* operand.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7 (on page 2493), *set* (on page 2553), *test*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215), **<limits.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 7

Austin Group Interpretations 1003.1-2001 #039, #040, and #094 are applied.

SD5-XCU-ERN-121 is applied, updating the EXAMPLES section.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0127 [291] is applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0150 [584] and XCU/TC2-2008/0151 [584] are applied.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

pax — portable archive interchange

SYNOPSIS

```
pax [-dv] [-c|-n] [-H|-L] [-o options] [-f archive] [-s replstr]...
    [pattern...]

pax -r[-c|-n] [-dikuv] [-H|-L] [-f archive] [-o options]... [-p string]...
    [-s replstr]... [pattern...]

pax -w [-dituvX] [-H|-L] [-b blocksize] [[-a] [-f archive]] [-o options]...
    [-s replstr]... [-x format] [file...]

pax -r -w [-dikltuvX] [-H|-L] [-o options]... [-p string]...
    [-s replstr]... [file...] directory
```

DESCRIPTION

The *pax* utility shall read, write, and write lists of the members of archive files and copy directory hierarchies. A variety of archive formats shall be supported; see the **–x** *format* option.

The action to be taken depends on the presence of the **–r** and **–w** options. The four combinations of **–r** and **–w** are referred to as the four modes of operation: **list**, **read**, **write**, and **copy** modes, corresponding respectively to the four forms shown in the SYNOPSIS section.

list In **list** mode (when neither **–r** nor **–w** are specified), *pax* shall write the names of the members of the archive file read from the standard input, with pathnames matching the specified patterns, to standard output. If a named file is of type directory, the file hierarchy rooted at that file shall be listed as well.

read In **read** mode (when **–r** is specified, but **–w** is not), *pax* shall extract the members of the archive file read from the standard input, with pathnames matching the specified patterns. If an extracted file is of type directory, the file hierarchy rooted at that file shall be extracted as well. The extracted files shall be created performing pathname resolution with the directory in which *pax* was invoked as the current working directory.

If an attempt is made to extract a directory when the directory already exists, this shall not be considered an error. If an attempt is made to extract a FIFO when the FIFO already exists, this shall not be considered an error.

The ownership, access, and modification times, and file mode of the restored files are discussed under the **–p** option.

write In **write** mode (when **–w** is specified, but **–r** is not), *pax* shall write the contents of the *file* operands to the standard output in an archive format. If no *file* operands are specified, a list of files to copy, one per line, shall be read from the standard input and each entry in this list shall be processed as if it had been a *file* operand on the command line. A file of type directory shall include all of the files in the file hierarchy rooted at the file.

copy In **copy** mode (when both **–r** and **–w** are specified), *pax* shall copy the *file* operands to the destination directory.

If no *file* operands are specified, a list of files to copy, one per line, shall be read from the standard input. A file of type directory shall include all of the files in the file hierarchy rooted at the file.

The effect of the **copy** shall be as if the copied files were written to a *pax* format archive file and then subsequently extracted, except that copying of sockets may be

supported even if archiving them in write mode is not supported, and that there may be hard links between the original and the copied files. If the destination directory is a subdirectory of one of the files to be copied, the results are unspecified. If the destination directory is a file of a type not defined by the System Interfaces volume of POSIX.1-2024, the results are implementation-defined; otherwise, it shall be an error for the file named by the *directory* operand not to exist, not be writable by the user, or not be a file of type directory.

In **read** or **copy** modes, if intermediate directories are necessary to extract an archive member, *pax* shall perform actions equivalent to the *mkdir*( ) function defined in the System Interfaces volume of POSIX.1-2024, called with the following arguments:

- The intermediate directory used as the *path* argument
- The value of the bitwise-inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO as the *mode* argument

If any specified *pattern* or *file* operands are not matched by at least one file or archive member, *pax* shall write a diagnostic message to standard error for each one that did not match and exit with a non-zero exit status.

The archive formats described in the EXTENDED DESCRIPTION section shall be automatically detected on input. The default output archive format shall be implementation-defined.

A single archive can span multiple files. The *pax* utility shall determine, in an implementation-defined manner, what file to read or write as the next file.

If the selected archive format supports the specification of linked files, it shall be an error if these files cannot be linked when the archive is extracted. For archive formats that do not store file contents with each name that causes a hard link, if the file that contains the data is not extracted during this *pax* session, either the data shall be restored from the original file, or a diagnostic message shall be displayed with the name of a file that can be used to extract the data. In traversing directories, *pax* shall detect infinite loops; that is, entering a previously visited directory that is an ancestor of the last file visited. When it detects an infinite loop, *pax* shall write a diagnostic message to standard error and shall terminate.

OPTIONS

The *pax* utility shall conform to XBD Section 12.2 (on page 215), except that the order of presentation of the **−o**, **−p**, and **−s** options is significant.

The following options shall be supported:

−r Read an archive file from standard input.

−w Write files to the standard output in the specified archive format.

−a Append files to the end of the archive. It is implementation-defined which devices on the system support appending. Additional file formats unspecified by this volume of POSIX.1-2024 may impose restrictions on appending.

−b *blocksize* Block the output at a positive decimal integer number of bytes per write to the archive file. Devices and archive formats may impose restrictions on blocking. Blocking shall be automatically determined on input. Conforming applications shall not specify a *blocksize* value larger than 32 256. Default blocking when creating archives depends on the archive format. (See the **−x** option below.)

−c Match all file or archive members except those specified by the *pattern* or *file* operands.

110065 **−d** Cause files of type directory being copied or archived or archive members of type
110066 directory being extracted or listed to match only the file or archive member itself
110067 and not the file hierarchy rooted at the file.

110068 **−f** *archive* Specify the pathname of the input or output archive, overriding the default
110069 standard input (in **list** or **read** modes) or standard output (**write** mode).

110070 **−H** If a symbolic link referencing a file of type directory is specified on the command
110071 line, *pax* shall archive the file hierarchy rooted in the file referenced by the link,
110072 using the name of the link as the root of the file hierarchy. Otherwise, if a symbolic
110073 link referencing a file of any other file type which *pax* can normally archive is
110074 specified on the command line, then *pax* shall archive the file referenced by the
110075 link, using the name of the link. The default behavior, when neither **−H** or **−L** are
110076 specified, shall be to archive the symbolic link itself.

110077 **−i** Interactively rename files or archive members. For each archive member matching
110078 a *pattern* operand or file matching a *file* operand, a prompt shall be written to the
110079 file **/dev/tty**. The prompt shall contain the name of the file or archive member, but
110080 the format is otherwise unspecified. A line shall then be read from **/dev/tty**. If this
110081 line is blank, the file or archive member shall be skipped. If this line consists of a
110082 single period, the file or archive member shall be processed with no modification
110083 to its name. Otherwise, its name shall be replaced with the contents of the line. The
110084 *pax* utility shall immediately exit with a non-zero exit status if end-of-file is
110085 encountered when reading a response or if **/dev/tty** cannot be opened for reading
110086 and writing.

110087 The results of extracting a hard link to a file that has been renamed during
110088 extraction are unspecified.

110089 **−k** Prevent the overwriting of existing files.

110090 **−l** (The letter ell.) In **copy** mode, hard links shall be made between the source and
110091 destination file hierarchies whenever possible. If specified in conjunction with **−H**
110092 or **−L**, when a symbolic link is encountered, the hard link created in the destination
110093 file hierarchy shall be to the file referenced by the symbolic link. If specified when
110094 neither **−H** nor **−L** is specified, when a symbolic link is encountered, the
110095 implementation shall create a hard link to the symbolic link in the source file
110096 hierarchy or copy the symbolic link to the destination.

110097 **−L** If a symbolic link referencing a file of type directory is specified on the command
110098 line or encountered during the traversal of a file hierarchy, *pax* shall archive the file
110099 hierarchy rooted in the file referenced by the link, using the name of the link as the
110100 root of the file hierarchy. Otherwise, if a symbolic link referencing a file of any
110101 other file type which *pax* can normally archive is specified on the command line or
110102 encountered during the traversal of a file hierarchy, *pax* shall archive the file
110103 referenced by the link, using the name of the link. The default behavior, when
110104 neither **−H** or **−L** are specified, shall be to archive the symbolic link itself.

110105 **−n** Select the first archive member that matches each *pattern* operand. No more than
110106 one archive member shall be matched for each pattern (although members of type
110107 directory shall still match the file hierarchy rooted at that file).

110108 **−o** *options* Provide information to the implementation to modify the algorithm for extracting
110109 or writing files. The value of *options* shall consist of one or more
110110 <comma>-separated keywords of the form:

110111 *keyword*`[[:]=`*value*`][,`*keyword*`[[:]=`*value*`], ...]`

Some keywords apply only to certain file formats, as indicated with each description. Use of keywords that are inapplicable to the file format being processed produces undefined results.

Keywords in the *options* argument shall be a string that would be a valid portable filename as described in XBD Section 3.265 (on page 70).

Note: Keywords are not expected to be filenames, merely to follow the same character composition rules as portable filenames.

Keywords can be preceded with white space. The *value* field shall consist of zero or more characters; within *value*, the application shall precede any literal <comma> with a <backslash>, which shall be ignored, but preserves the <comma> as part of *value*. A <comma> as the final character, or a <comma> followed solely by white space as the final characters, in *options* shall be ignored. Multiple **–o** options can be specified; if keywords given to these multiple **–o** options conflict, the keywords and values appearing later in command line sequence shall take precedence and the earlier shall be silently ignored. The following keyword values of *options* shall be supported for the file formats as indicated:

delete=*pattern*
(Applicable only to the **–x pax** format.) When used in **write** or **copy** mode, *pax* shall omit from extended header records that it produces any keywords matching the string pattern. When used in **read** or **list** mode, *pax* shall ignore any keywords matching the string pattern in the extended header records. In both cases, matching shall be performed using the pattern matching notation described in Section 2.14.1 (on page 2523) and Section 2.14.2 (on page 2524). For example:

`-o` **delete**=*`security.*`*

would suppress security-related information. See pax Extended Header (on page 3264) for extended header record keyword usage.

When multiple **–odelete=pattern** options are specified, the patterns shall be additive; all keywords matching the specified string patterns shall be omitted from extended header records that *pax* produces.

exthdr.name=*string*
(Applicable only to the **–x pax** format.) This keyword allows user control over the name that is written into the **ustar** header blocks for the extended header produced under the circumstances described in pax Header Block (on page 3263). The name shall be the contents of *string*, after the following character substitutions have been made:

string **Includes:**	**Replaced by:**
`%d`	The directory name of the file, equivalent to the result of the *dirname* utility on the translated pathname.
`%f`	The filename of the file, equivalent to the result of the *basename* utility on the translated pathname.
`%p`	The process ID of the *pax* process.
`%%`	A `'%'` character.

Any other `'%'` characters in *string* produce undefined results.

If no **–o exthdr.name=string** is specified, *pax* shall use the following default

value:

```
%d/PaxHeaders.%p/%f
```

globexthdr.name=*string*

(Applicable only to the –**x pax** format.) When used in **write** or **copy** mode with the appropriate options, *pax* shall create global extended header records with **ustar** header blocks that are treated as regular files by previous versions of *pax*. This keyword allows user control over the name that is written into the **ustar** header blocks for global extended header records. The name shall be the contents of string, after the following character substitutions have been made:

string **Includes:**	**Replaced by:**
`%n`	An integer that represents the sequence number of the global extended header record in the archive, starting at 1.
`%p`	The process ID of the *pax* process.
`%%`	A `'%'` character.

Any other `'%'` characters in *string* produce undefined results.

If no –**o globexthdr.name=string** is specified, *pax* shall use the following default value:

```
$TMPDIR/GlobalHead.%p.%n
```

where $*TMPDIR* represents the value of the *TMPDIR* environment variable. If *TMPDIR* is not set, *pax* shall use **/tmp**.

invalid=*action*

(Applicable only to the –**x pax** format.) This keyword allows user control over the action *pax* takes upon encountering values in an extended header record that, in **read** or **copy** mode, are invalid in the destination hierarchy or, in **list** mode, cannot be written in the codeset and current locale of the implementation. The following are invalid values that shall be recognized by *pax*:

— In **read** or **copy** mode, a filename or link name that contains character encodings invalid in the destination hierarchy. (For example, the name may contain embedded NULs.)

— In **read** or **copy** mode, a filename or link name that is longer than the maximum allowed in the destination hierarchy (for either a pathname component or the entire pathname).

— In **list** mode, any character string value (filename, link name, user name, and so on) that cannot be written in the codeset and current locale of the implementation.

The following mutually-exclusive values of the *action* argument are supported:

binary In **write** mode, *pax* shall generate a **hdrcharset**=**BINARY** extended header record for each file with a filename, link name, group name, owner name, or any other field in an extended header record that cannot be translated to the UTF-8 codeset, allowing the archive to contain the files with unencoded extended header record values. In **read** or **copy** mode, *pax* shall use the values specified in the header without translation,

regardless of whether this may overwrite an existing file with a valid name. In **list** mode, *pax* shall behave identically to the **bypass** action.

bypass In **read** or **copy** mode, *pax* shall bypass the file, causing no change to the destination hierarchy. In **list** mode, *pax* shall write all requested valid values for the file, but its method for writing invalid values is unspecified.

rename In **read** or **copy** mode, *pax* shall act as if the –**i** option were in effect for each file with invalid filename or link name values, allowing the user to provide a replacement name interactively. In **list** mode, *pax* shall behave identically to the **bypass** action.

UTF-8 When used in **read**, **copy**, or **list** mode and a filename, link name, owner name, or any other field in an extended header record cannot be translated from the **pax** UTF-8 codeset format to the codeset and current locale of the implementation, *pax* shall use the actual UTF-8 encoding for the name. If a **hdrcharset** extended header record is in effect for this file, the character set specified by that record shall be used instead of UTF-8. If a **hdrcharset**=**BINARY** extended header record is in effect for this file, no translation shall be performed.

write In **read** or **copy** mode, *pax* shall write the file, translating the name, regardless of whether this may overwrite an existing file with a valid name. In **list** mode, *pax* shall behave identically to the **bypass** action.

If no –**o invalid=option** is specified, *pax* shall act as if –**oinvalid=bypass** were specified. Any overwriting of existing files that may be allowed by the –**oinvalid=** actions shall be subject to permission (–**p**) and modification time (–**u**) restrictions, and shall be suppressed if the –**k** option is also specified.

linkdata
(Applicable only to the –**x pax** format.) In **write** mode, *pax* shall write the contents of a file to the archive even when that file is merely a hard link to a file whose contents have already been written to the archive.

listopt=*format*
This keyword specifies the output format of the table of contents produced when the –**v** option is specified in **list** mode. See List Mode Format Specifications (on page 3258). To avoid ambiguity, the **listopt=format** shall be the only or final **keyword=value** pair in a –**o** option-argument; all characters in the remainder of the option-argument shall be considered part of the format string. When multiple –**olistopt=format** options are specified, the format strings shall be considered a single, concatenated string, evaluated in command line order.

times
(Applicable only to the –**x** *pax* format.) When used in **write** or **copy** mode, *pax* shall include **atime** and **mtime** extended header records for each file. See pax Extended Header File Times (on page 3267).

In addition to these keywords, if the –**x** *pax* format is specified, any of the keywords and values defined in pax Extended Header (on page 3264), including

110250 implementation extensions, can be used in −**o** option-arguments, in either of two
110251 modes:

110252 **keyword**=*value*
110253 When used in **write** or **copy** mode, these keyword/value pairs shall be
110254 included at the beginning of the archive as **typeflag g** global extended header
110255 records. When used in **read** or **list** mode, these keyword/value pairs shall act
110256 as if they had been at the beginning of the archive as **typeflag g** global
110257 extended header records.

110258 **keyword**:=*value*
110259 When used in **write** or **copy** mode, these keyword/value pairs shall be
110260 included as records at the beginning of a **typeflag x** extended header for each
110261 file. (This shall be equivalent to the **<**equals-sign**>** form except that it creates
110262 no **typeflag g** global extended header records.) When used in **read** or **list**
110263 mode, these keyword/value pairs shall act as if they were included as records
110264 at the end of each extended header; thus, they shall override any global or file-
110265 specific extended header record keywords of the same names. For example, in
110266 the command:

110267 ```
110268 pax -r -o "
110269 gname:=mygroup,
" <archive
```

110270 the group name is forced to a new value for all files read from the archive.

110271 The precedence of −**o** keywords over various fields in the archive is described in
110272 pax Extended Header Keyword Precedence (on page 3267). If the −**o**
110273 **delete**=*pattern*, −**o keyword**=*value*, or −**o keyword**:=*value* options are used to
110274 override or remove any extended header data needed to find files in an archive
110275 (e.g., `-o delete=size` for a file whose size cannot be represented in a **ustar**
110276 header or `-o size=100` for a file whose size is not 100 bytes), the behavior is
110277 undefined.

110278 −**p** *string* Specify one or more file characteristic options (privileges). The *string* option-
110279 argument shall be a string specifying file characteristics to be retained or discarded
110280 on extraction. The string shall consist of the specification characters `a`, `e`, `m`, `o`, and
110281 `p`. Other implementation-defined characters can be included. Multiple
110282 characteristics can be concatenated within the same string and multiple −**p** options
110283 can be specified. The meaning of the specification characters are as follows:

110284 `a` Do not preserve file access times.

110285 `e` Preserve the user ID, group ID, file mode bits (see XBD Section 3.145, on page
110286 52), access time, modification time, and any other implementation-defined file
110287 characteristics.

110288 `m` Do not preserve file modification times.

110289 `o` Preserve the user ID and group ID.

110290 `p` Preserve the file mode bits. Other implementation-defined file mode attributes
110291 may be preserved.

110292 In the preceding list, ``preserve'' indicates that an attribute stored in the archive
110293 shall be given to the extracted file, subject to the permissions of the invoking
110294 process. The access and modification times of the file shall be preserved unless
110295 otherwise specified with the −**p** option or not stored in the archive. All attributes

  
```

that are not preserved shall be determined as part of the normal file creation action (see Section 1.1.1.4, on page 2454).

If neither the `e` nor the `o` specification character is specified, or the user ID and group ID are not preserved for any reason, *pax* shall not set the S_ISUID and S_ISGID bits of the file mode.

If the preservation of any of these items fails for any reason, *pax* shall write a diagnostic message to standard error. Failure to preserve these items shall affect the final exit status, but shall not cause the extracted file to be deleted.

If file characteristic letters in any of the *string* option-arguments are duplicated or conflict with each other, the ones given last shall take precedence. For example, if **−p** `eme` is specified, file modification times are preserved.

−s *replstr* Modify file or archive member names named by *pattern* or *file* operands according to the substitution expression *replstr*, using the syntax of the *ed* utility. The concepts of ``address'' and ``line'' are meaningless in the context of the *pax* utility, and shall not be supplied. The format shall be:

`-s /`*`old`*`/`*`new`*`/`**`[`**`gpsS`**`]`**

where as in *ed*, *old* is a basic regular expression and *new* can contain an **<**ampersand**>**, `'\n'` (where *n* is a digit) back-references, or subexpression matching. The *old* string shall also be permitted to contain **<**newline**>** characters.

Any non-null character can be used as a delimiter (`'/'` shown here). Multiple **−s** expressions can be specified; the expressions shall be applied in the order specified, terminating with the first successful substitution. The optional trailing `'g'` is as defined in the *ed* utility. The optional trailing `'p'` shall cause successful substitutions to be written to standard error. The optional trailing `'s'` and `'S'` control whether the substitutions are applied to symbolic link contents: `'s'` shall cause them not to be applied; `'S'` shall cause them to be applied. If neither is present, it is unspecified which is the default. If both are present, the behavior is unspecified. File or archive member names that substitute to the empty string shall be ignored when reading and writing archives. Symbolic link contents that substitute to the empty string shall not be treated specially.

−t When reading files from the file system, and if the user has the permissions required by *futimens*() to do so, set the access time of each file read to the access time that it had before being read by *pax*.

−u Ignore files that are older (having a less recent file modification time) than a pre-existing file or archive member with the same name. In **read** mode, an archive member with the same name as a file in the file system shall be extracted if the archive member is newer than the file. In **write** mode, an archive file member with the same name as a file in the file system shall be superseded if the file is newer than the archive member. If **−a** is also specified, this is accomplished by appending to the archive; otherwise, it is unspecified whether this is accomplished by actual replacement in the archive or by appending to the archive. In **copy** mode, the file in the destination hierarchy shall be replaced if the file in the source hierarchy is newer.

−v In **list** mode, produce a verbose table of contents (see the STDOUT section). Otherwise, write archive member pathnames to standard error (see the STDERR section).

110342 –**x** *format* Specify the output archive format. The *pax* utility shall support the following
110343 formats:

110344 **cpio** The **cpio** interchange format; see the EXTENDED DESCRIPTION
110345 section. The default *blocksize* for this format for character special
110346 archive files shall be 5 120. Implementations shall support all
110347 *blocksize* values less than or equal to 32 256 that are multiples of 512.

110348 **pax** The **pax** interchange format; see the EXTENDED DESCRIPTION
110349 section. The default *blocksize* for this format for character special
110350 archive files shall be 5 120. Implementations shall support all
110351 *blocksize* values less than or equal to 32 256 that are multiples of 512.

110352 **ustar** The **tar** interchange format; see the EXTENDED DESCRIPTION
110353 section. The default *blocksize* for this format for character special
110354 archive files shall be 10 240. Implementations shall support all
110355 *blocksize* values less than or equal to 32 256 that are multiples of 512.

110356 Implementation-defined formats shall specify a default block size as well as any
110357 other block sizes supported for character special archive files.

110358 Any attempt to append to an archive file in a format different from the existing
110359 archive format shall cause *pax* to exit immediately with a non-zero exit status.

110360 –**X** When traversing the file hierarchy specified by a pathname, *pax* shall not descend
110361 below directories that have a different device ID (*st_dev*; see XSH *fstatat*( )) than the
110362 specified pathname; that is, when a directory with a different device ID is
110363 encountered, *pax* shall process (archive or copy) the directory itself but shall not
110364 process any files below the directory.

110365 Specifying more than one of the mutually-exclusive options –**H** and –**L** shall not be considered
110366 an error and the last option specified shall determine the behavior of the utility.

110367 The options that operate on the names of files or archive members (–**c**, –**i**, –**n**, –**s**, –**u**, and –**v**)
110368 shall interact as follows. In **read** mode, the archive members shall be selected based on the user-
110369 specified *pattern* operands as modified by the –**c**, –**n**, and –**u** options. Then, any –**s** and –**i**
110370 options shall modify, in that order, the names of the selected files. The –**v** option shall write
110371 names resulting from these modifications.

110372 In **write** mode, the files shall be selected based on the user-specified pathnames as modified by
110373 the –**u** option. Then, any –**s** and –**i** options shall modify, in that order, the names of these
110374 selected files. The –**v** option shall write names resulting from these modifications.

110375 If both the –**u** and –**n** options are specified, *pax* shall not consider a file selected unless it is
110376 newer than the file to which it is compared.

110377 **List Mode Format Specifications**

110378 In **list** mode with the –**o listopt=format** option, the *format* argument shall be applied for each
110379 selected file. The *pax* utility shall append a **<**newline**>** to the **listopt** output for each selected file.
110380 The *format* argument shall be used as the *format* string described in XBD Chapter 5 (on page 113),
110381 with the exceptions 1. through 6. defined in the EXTENDED DESCRIPTION section of *printf*,
110382 plus the following exceptions:

110383 7. The sequence (*keyword*) can occur before a format conversion specifier. The conversion
110384 argument is defined by the value of *keyword*. The implementation shall support the
110385 following keywords:

110386 — Any of the Field Name entries in Table 3-15 (on page 3268) and Table 3-17 (on page
110387 3272). The implementation may support the *cpio* keywords without the leading **c_** in
110388 addition to the form required by Table 3-17 (on page 3272).

110389 — Any keyword defined for the extended header in pax Extended Header (on page
110390 3264).

110391 — Any keyword provided as an implementation-defined extension within the extended
110392 header defined in pax Extended Header (on page 3264).

110393 For example, the sequence `"%(charset)s"` is the string value of the name of the character
110394 set in the extended header.

110395 The result of the keyword conversion argument shall be the value from the applicable
110396 header field or extended header, without any trailing NULs.

110397 All keyword values used as conversion arguments shall be translated from the UTF-8
110398 encoding (or alternative encoding specified by any **hdrcharset** extended header record) to
110399 the character set appropriate for the local file system, user database, and so on, as
110400 applicable.

110401 8. An additional conversion specifier character, `T`, shall be used to specify time formats. The `T`
110402 conversion specifier character can be preceded by the sequence (*keyword*=*subformat*), where
110403 *subformat* is a date format as defined by *date* operands. The default *keyword* shall be **mtime**
110404 and the default subformat shall be:

110405 `%b %e %H:%M %Y`

110406 9. An additional conversion specifier character, `M`, shall be used to specify the file mode string
110407 as defined in *ls* Standard Output. If (*keyword*) is omitted, the **mode** keyword shall be used.
110408 For example, `%.1M` writes the single character corresponding to the <*entry type*> field of the
110409 *ls* **–l** command.

110410 10. An additional conversion specifier character, `D`, shall be used to specify the device for block
110411 or special files, if applicable, in an implementation-defined format. If not applicable, and
110412 (*keyword*) is specified, then this conversion shall be equivalent to `%(`*keyword*`)u`. If not
110413 applicable, and (*keyword*) is omitted, then this conversion shall be equivalent to <space>.

110414 11. An additional conversion specifier character, `F`, shall be used to specify a pathname. The `F`
110415 conversion character can be preceded by a sequence of <comma>-separated keywords:

110416 `(`*keyword*`[,`*keyword*`] ... )`

110417 The values for all the keywords that are non-null shall be concatenated together, each
110418 separated by a `'/'`. The default shall be (**path**) if the keyword **path** is defined; otherwise,
110419 the default shall be (**prefix**,**name**).

110420 12. An additional conversion specifier character, `L`, shall be used to specify a symbolic link
110421 expansion. If the current file is a symbolic link, then `%L` shall expand to:

110422 `"%s -> %s", <`*value of keyword*`>, <`*contents of link*`>`

110423 Otherwise, the `%L` conversion specification shall be the equivalent of `%F`.

110424 **OPERANDS**

110425 The following operands shall be supported:

110426 *directory* The destination directory pathname for **copy** mode.

file A pathname of a file to be copied or archived.

pattern A pattern matching one or more pathnames of archive members. A pattern needs to be given in the name-generating notation of the pattern matching notation in Section 2.14 (on page 2523), including the filename expansion rules in Section 2.14.3 (on page 2525). The default, if no *pattern* is specified, is to select all members in the archive.

STDIN

In **write** mode, the standard input shall be used only if no *file* operands are specified. It shall be a file containing a list of pathnames, each terminated by a <newline> character.

In **list** and **read** modes, if **−f** is not specified, the standard input shall be an archive file.

Otherwise, the standard input shall not be used.

INPUT FILES

The input file named by the *archive* option-argument, or standard input when the archive is read from there, shall be a file formatted according to one of the specifications in the EXTENDED DESCRIPTION section or some other implementation-defined format.

The file **/dev/tty** shall be used to write prompts and read responses.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *pax*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements used in the pattern matching expressions for the *pattern* operand and the basic regular expression for the **−s** option.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), and the behavior of character classes used in the pattern matching expressions for the *pattern* operand and the basic regular expression for the **−s** option.

LC_MESSAGES
Determine the locale used to affect the format and contents of diagnostic messages and prompts written to standard error.

LC_TIME Determine the format and contents of date and time strings when the **−v** option is specified.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

TMPDIR Determine the pathname that provides part of the default global extended header record file, as described for the **−o globexthdr=** keyword in the OPTIONS section.

TZ Determine the timezone used to calculate date and time strings when the **−v** option is specified. If *TZ* is unset or null, an unspecified default timezone shall be used.

ASYNCHRONOUS EVENTS

Default.

STDOUT

In **write** mode, if −**f** is not specified, the standard output shall be the archive formatted according to one of the specifications in the EXTENDED DESCRIPTION section, or some other implementation-defined format (see −**x** *format*).

In **list** mode, when the −**olistopt**=*format* has been specified, the selected archive members shall be written to standard output using the format described under List Mode Format Specifications (on page 3258). In **list** mode without the −**olistopt**=*format* option, the table of contents of the selected archive members shall be written to standard output using the following format:

```
"%s\n", <pathname>
```

If the −**v** option is specified in **list** mode, the table of contents of the selected archive members shall be written to standard output using the following formats.

For pathnames representing hard links to previous members of the archive:

```
"%sΔ==Δ%s\n", <ls −l listing>, <linkname>
```

For all other pathnames:

```
"%s\n", <ls −l listing>
```

where **<***ls* −l *listing***>** shall be the format specified by the *ls* utility with the −**l** option. When writing pathnames in this format, it is unspecified what is written for fields for which the underlying archive format does not have the correct information, although the correct number of **<**blank**>**-separated fields shall be written.

In **list** mode, standard output shall not be buffered more than a pathname (plus any associated information and a **<**newline**>** terminator) at a time.

STDERR

If −**v** is specified in **read**, **write**, or **copy** modes, *pax* shall write the pathnames it processes to the standard error output using the following format:

```
"%s\n", <pathname>
```

These pathnames shall be written as soon as processing is begun on the file or archive member, and shall be flushed to standard error. The trailing **<**newline**>**, which shall not be buffered, is written when the file has been read or written.

If the −**s** option is specified, and the replacement string has a trailing `'p'`, substitutions shall be written to standard error in the following format:

```
"%sΔ>>Δ%s\n", <original pathname>, <new pathname>
```

In all operating modes of *pax*, optional messages of unspecified format concerning the input archive format and volume number, the number of files, blocks, volumes, and media parts as well as other diagnostic messages may be written to standard error.

In all formats, for both standard output and standard error, it is unspecified how non-printable characters in pathnames or link names are written.

When using the −**xpax** archive format, if a filename, link name, group name, owner name, or any other field in an extended header record cannot be translated between the codeset in use for that extended header record and the character set of the current locale, *pax* shall write a diagnostic message to standard error, shall process the file as described for the −**o invalid=** option, and then shall continue processing with the next file.

OUTPUT FILES

In **read** mode, the extracted output files shall be of the archived file type. In **copy** mode, the copied output files shall be the type of the file being copied. In either mode, existing files in the destination hierarchy shall be overwritten only when all permission (**–p**), modification time (**–u**), and invalid-value (**–oinvalid=**) tests allow it.

In **write** mode, the output file named by the **–f** option-argument shall be a file formatted according to one of the specifications in the EXTENDED DESCRIPTION section, or some other implementation-defined format.

EXTENDED DESCRIPTION

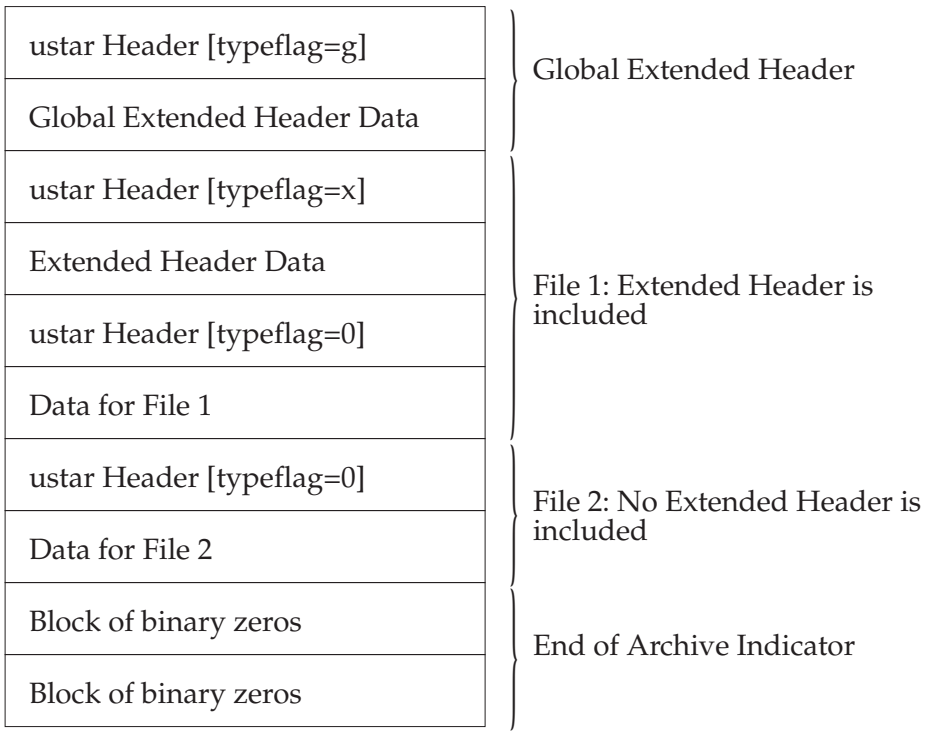
pax Interchange Format

A *pax* archive tape or file produced in the **–xpax** format shall contain a series of blocks. The physical layout of the archive shall be identical to the **ustar** format described in ustar Interchange Format (on page 3268). Each file archived shall be represented by the following sequence:

- An optional header block with extended header records. This header block is of the form described in pax Header Block (on page 3263), with a *typeflag* value of **x** or **g**. The extended header records, described in pax Extended Header (on page 3264), shall be included as the data for this header block.
- A header block that describes the file. Any fields in the preceding optional extended header shall override the associated fields in this header block for this file.
- Zero or more blocks that contain the contents of the file.

At the end of the archive file there shall be two 512-byte blocks filled with binary zeros, interpreted as an end-of-archive indicator.

A schematic of an example archive with global extended header records and two actual files is shown in Figure 3-1 (on page 3263). In the example, the second file in the archive has no extended header preceding it, presumably because it has no need for extended attributes.



110538 **Figure 3-1** pax Format Archive Example

110539 **pax Header Block**

110540 The **pax** header block shall be identical to the **ustar** header block described in ustar Interchange
110541 Format (on page 3268), except that two additional *typeflag* values are defined:

110542 x Represents extended header records for the following file in the archive (which shall have
110543 its own **ustar** header block). The format of these extended header records shall be as
110544 described in pax Extended Header (on page 3264).

110545 g Represents global extended header records for the following files in the archive. The format
110546 of these extended header records shall be as described in pax Extended Header (on page
110547 3264). Each value shall affect all subsequent files that do not override that value in their
110548 own extended header record and until another global extended header record is reached
110549 that provides another value for the same field. The *typeflag* **g** global headers should not be
110550 used with interchange media that could suffer partial data loss in transporting the archive.

110551 For both of these types, the *size* field shall be the size of the extended header records in octets.
110552 The other fields in the header block are not meaningful to this version of the *pax* utility.
110553 However, if this archive is read by a *pax* utility conforming to the ISO POSIX-2: 1993 standard,
110554 the header block fields are used to create a regular file that contains the extended header records
110555 as data. Therefore, header block field values should be selected to provide reasonable file access
110556 to this regular file.

110557 A further difference from the **ustar** header block is that data blocks for files of *typeflag* 1 (the digit
110558 one) (hard link) may be included, which means that the size field may be greater than zero.
110559 Archives created by *pax* **–o linkdata** shall include these data blocks with the hard links.

110560 **pax Extended Header**

110561 A **pax** extended header contains values that are inappropriate for the **ustar** header block because
110562 of limitations in that format: fields requiring a character encoding other than that described in
110563 the ISO/IEC 646:1991 standard, fields representing file attributes not described in the **ustar**
110564 header, and fields whose format or length do not fit the requirements of the **ustar** header. The
110565 values in an extended header add attributes to the following file (or files; see the description of
110566 the *typeflag* **g** header block) or override values in the following header block(s), as indicated in
110567 the following list of keywords.

110568 An extended header shall consist of one or more records, each constructed as follows:

110569 `"%d %s=%s\n", <length>, <keyword>, <value>`

110570 The extended header records shall be encoded according to the ISO/IEC 10646:2020 standard
110571 UTF-8 encoding. The <*length*> field, <blank>, <equals-sign>, and <newline> shown shall be
110572 limited to the portable character set, as encoded in UTF-8. The <*keyword*> fields can be any
110573 UTF-8 characters. The <*length*> field shall be the decimal length of the extended header record
110574 in octets, including the trailing <newline>. If there is a **hdrcharset** extended header in effect for
110575 a file, the *value* field for any **gname**, **linkpath**, **path**, and **uname** extended header records shall be
110576 encoded using the character set specified by the **hdrcharset** extended header record; otherwise,
110577 the *value* field shall be encoded using UTF-8. The *value* field for all other keywords specified by
110578 POSIX.1-2024 shall be encoded using UTF-8.

110579 The <*keyword*> field shall be one of the entries from the following list or a keyword provided as
110580 an implementation extension. Keywords consisting entirely of lowercase letters, digits, and
110581 periods are reserved for future standardization. A keyword shall not include an <equals-sign>.
110582 (In the following list, the notations ``file(s)'' or ``block(s)'' is used to acknowledge that a keyword
110583 affects the following single file after a *typeflag* **x** extended header, but possibly multiple files after
110584 *typeflag* **g**. Any requirements in the list for *pax* to include a record when in **write** or **copy** mode
110585 shall apply only when such a record has not already been provided through the use of the **–o**
110586 option. When used in **copy** mode, *pax* shall behave as if an archive had been created with
110587 applicable extended header records and then extracted.)

110588 **atime** The file access time for the following file(s), equivalent to the value of the *st_atim*
110589 member of the **stat** structure for a file, as described by the *stat*( ) function. The
110590 access time shall be restored if the process has appropriate privileges required to
110591 do so. The format of the <*value*> shall be as described in pax Extended Header File
110592 Times (on page 3267).

110593 **charset** The name of the character set used to encode the data in the following file(s). The
110594 entries in the following table are defined to refer to known standards; additional
110595 names may be agreed on between the originator and recipient.

<value>	Formal Standard
ISO-IR∆646∆1990	ISO/IEC 646: 1990
ISO-IR∆8859∆1∆1998	ISO/IEC 8859-1: 1998
ISO-IR∆8859∆2∆1999	ISO/IEC 8859-2: 1999
ISO-IR∆8859∆3∆1999	ISO/IEC 8859-3: 1999
ISO-IR∆8859∆4∆1998	ISO/IEC 8859-4: 1998
ISO-IR∆8859∆5∆1999	ISO/IEC 8859-5: 1999
ISO-IR∆8859∆6∆1999	ISO/IEC 8859-6: 1999
ISO-IR∆8859∆7∆1987	ISO/IEC 8859-7: 1987
ISO-IR∆8859∆8∆1999	ISO/IEC 8859-8: 1999
ISO-IR∆8859∆9∆1999	ISO/IEC 8859-9: 1999
ISO-IR∆8859∆10∆1998	ISO/IEC 8859-10: 1998
ISO-IR∆8859∆13∆1998	ISO/IEC 8859-13: 1998
ISO-IR∆8859∆14∆1998	ISO/IEC 8859-14: 1998
ISO-IR∆8859∆15∆1999	ISO/IEC 8859-15: 1999
ISO-IR∆10646∆2000	ISO/IEC 10646: 2000
ISO-IR∆10646∆2000∆UTF-8	ISO/IEC 10646, UTF-8 encoding
BINARY	None.

The encoding is included in an extended header for information only; when *pax* is used as described in POSIX.1-2024, it shall not translate the file data into any other encoding. The **BINARY** entry indicates unencoded binary data.

When used in **write** or **copy** mode, it is implementation-defined whether *pax* includes a **charset** extended header record for a file.

comment A series of characters used as a comment. All characters in the <*value*> field shall be ignored by *pax*.

gid The group ID of the group that owns the file, expressed as a decimal number using digits from the ISO/IEC 646: 1991 standard. This record shall override the *gid* field in the following header block(s). When used in **write** or **copy** mode, *pax* shall include a *gid* extended header record for each file whose group ID is greater than 2 097 151 (octal 7 777 777).

gname The group of the file(s), formatted as a group name in the group database. This record shall override the *gid* and *gname* fields in the following header block(s), and any *gid* extended header record. When used in **read**, **copy**, or **list** mode, *pax* shall translate the name from the encoding in the header record to the character set appropriate for the group database on the receiving system. If any of the characters cannot be translated, and if neither the **−oinvalid=UTF-8** option nor the **−oinvalid=binary** option is specified, the results are implementation-defined. When used in **write** or **copy** mode, *pax* shall include a **gname** extended header record for each file whose group name cannot be represented entirely with the letters and digits of the portable character set.

hdrcharset The name of the character set used to encode the value field of the **gname**, **linkpath**, **path**, and **uname** *pax* extended header records. The entries in the following table are defined to refer to known standards; additional names may be agreed between the originator and the recipient.

<value>	Formal Standard
ISO-IRΔ10646Δ2000ΔUTF-8	ISO/IEC 10646, UTF-8 encoding
BINARY	None.

If no **hdrcharset** extended header record is specified, the default character set used to encode all values in extended header records shall be the ISO/IEC 10646:2020 standard UTF-8 encoding.

The **BINARY** entry indicates that all values recorded in extended headers for affected files are unencoded binary data from the underlying system.

linkpath The pathname of a link being created to another file, of any type, previously archived. This record shall override the *linkname* field in the following **ustar** header block(s). The following **ustar** header block shall determine the type of link created. If *typeflag* of the following header block is 1, it shall be a hard link. If *typeflag* is 2, it shall be a symbolic link and the **linkpath** value shall be the contents of the symbolic link. The *pax* utility shall translate the name of the link (contents of the symbolic link) from the encoding in the header to the character set appropriate for the local file system. When used in **write** or **copy** mode, *pax* shall include a **linkpath** extended header record for each link whose pathname cannot be represented entirely with the members of the portable character set other than NUL.

mtime The file modification time of the following file(s), equivalent to the value of the *st_mtim* member of the **stat** structure for a file, as described in the *stat*( ) function. This record shall override the *mtime* field in the following header block(s). The modification time shall be restored if the process has appropriate privileges required to do so. The format of the <*value*> shall be as described in pax Extended Header File Times (on page 3267).

path The pathname of the following file(s). This record shall override the *name* and *prefix* fields in the following header block(s). The *pax* utility shall translate the pathname of the file from the encoding in the header to the character set appropriate for the local file system.

When used in **write** or **copy** mode, *pax* shall include a *path* extended header record for each file whose pathname cannot be represented entirely with the members of the portable character set other than NUL.

realtime.*any* The keywords prefixed by ``realtime.'' are reserved for future standardization.

security.*any* The keywords prefixed by ``security.'' are reserved for future standardization.

size The size of the file in octets, expressed as a decimal number using digits from the ISO/IEC 646:1991 standard. This record shall override the *size* field in the following header block(s). When used in **write** or **copy** mode, *pax* shall include a *size* extended header record for each file with a size value greater than 8 589 934 591 (octal 77 777 777 777).

uid The user ID of the file owner, expressed as a decimal number using digits from the ISO/IEC 646:1991 standard. This record shall override the *uid* field in the following header block(s). When used in **write** or **copy** mode, *pax* shall include a *uid* extended header record for each file whose owner ID is greater than 2 097 151 (octal 7 777 777).

uname The owner of the following file(s), formatted as a user name in the user database. This record shall override the *uid* and *uname* fields in the following header block(s), and any *uid* extended header record. When used in **read**, **copy**, or **list** mode, *pax* shall translate the name from the encoding in the header record to the character set appropriate for the user database on the receiving system. If any of the characters cannot be translated, and if neither the −**oinvalid=UTF-8** option nor the −**oinvalid=binary** option is specified, the results are implementation-defined. When used in **write** or **copy** mode, *pax* shall include a **uname** extended header record for each file whose user name cannot be represented entirely with the letters and digits of the portable character set.

If the <*value*> field is zero length, it shall delete any header block field, previously entered extended header value, or global extended header value of the same name.

If a keyword in an extended header record (or in a −**o** option-argument) overrides or deletes a corresponding field in the **ustar** header block, *pax* shall ignore the contents of that header block field.

Unlike the **ustar** header block fields, NULs shall not delimit <*value*>s; all characters within the <*value*> field shall be considered data for the field. None of the length limitations of the **ustar** header block fields in Table 3-15 (on page 3268) shall apply to the extended header records.

pax Extended Header Keyword Precedence

This section describes the precedence in which the various header records and fields and command line options are selected to apply to a file in the archive. When *pax* is used in **read** or **list** modes, it shall determine a file attribute in the following sequence:

1. If −**odelete=keyword-prefix** is used, the affected attributes shall be determined from step 7., if applicable, or ignored otherwise.
2. If −**o***keyword*:= is used, the affected attributes shall be ignored.
3. If −**okeyword:=value** is used, the affected attribute shall be assigned the value.
4. If there is a *typeflag* **x** extended header record, the affected attribute shall be assigned the <*value*>. When extended header records conflict, the last one given in the header shall take precedence.
5. If −**okeyword=value** is used, the affected attribute shall be assigned the value.
6. If there is a *typeflag* **g** global extended header record, the affected attribute shall be assigned the <*value*>. When global extended header records conflict, the last one given in the global header shall take precedence.
7. Otherwise, the attribute shall be determined from the **ustar** header block.

pax Extended Header File Times

The *pax* utility shall write an **mtime** record for each file in **write** or **copy** modes if the file's modification time cannot be represented exactly in the **ustar** header logical record described in ustar Interchange Format (on page 3268). This can occur if the time is out of **ustar** range, or if the file system of the underlying implementation supports non-integer time granularities and the time is not an integer. All of these time records shall be formatted as a decimal representation of the time in seconds since the Epoch. If a <period> (`'.'`) decimal point character is present, the digits to the right of the point shall represent the units of a subsecond timing granularity, where the first digit is tenths of a second and each subsequent digit is a tenth of the previous digit. In **read** or **copy** mode, the *pax* utility shall truncate the time of a file to the greatest value

that is not greater than the input header file time. In **write** or **copy** mode, the *pax* utility shall output a time exactly if it can be represented exactly as a decimal number, and otherwise shall generate only enough digits so that the same time shall be recovered if the file is extracted on a system whose underlying implementation supports the same time granularity.

ustar Interchange Format

A **ustar** archive tape or file shall contain a series of logical records. Each logical record shall be a fixed-size logical record of 512 octets (see below). Although this format may be thought of as being stored on 9-track industry-standard 12.7 mm (0.5 in) magnetic tape, other types of transportable media are not excluded. Each file archived shall be represented by a header logical record that describes the file, followed by zero or more logical records that give the contents of the file. At the end of the archive file there shall be two 512-octet logical records filled with binary zeros, interpreted as an end-of-archive indicator.

The logical records may be grouped for physical I/O operations, as described under the **−b***blocksize* and **−x ustar** options. Each group of logical records may be written with a single operation equivalent to the *write*( ) function. On magnetic tape, the result of this write shall be a single tape physical block. The last physical block shall always be the full size, so logical records after the two zero logical records may contain undefined data.

The header logical record shall be structured as shown in the following table. All lengths and offsets are in decimal.

Table 3-15 ustar Header Block

Field Name	Octet Offset	Length (in Octets)
name	0	100
mode	100	8
uid	108	8
gid	116	8
size	124	12
mtime	136	12
chksum	148	8
typeflag	156	1
linkname	157	100
magic	257	6
version	263	2
uname	265	32
gname	297	32
devmajor	329	8
devminor	337	8
prefix	345	155

All characters in the header logical record shall be represented in the coded character set of the ISO/IEC 646: 1991 standard. For maximum portability between implementations, names should be selected from characters represented by the portable filename character set as octets with the most significant bit zero. If an implementation supports the use of characters outside of <slash> and the portable filename character set in names for files, users, and groups, one or more implementation-defined encodings of these characters shall be provided for interchange purposes.

However, the *pax* utility shall never create filenames on the local system that cannot be accessed

via the procedures described in POSIX.1-2024. If a filename is found on the medium that would create an invalid filename, it is implementation-defined whether the data from the file is stored on the file hierarchy and under what name it is stored. The *pax* utility may choose to ignore these files as long as it produces an error indicating that the file is being ignored.

Each field within the header logical record is contiguous; that is, there is no padding used. Each character on the archive medium shall be stored contiguously.

The fields *magic*, *uname*, and *gname* are character strings each terminated by a NUL character. The fields *name*, *linkname*, and *prefix* are NUL-terminated character strings except when all characters in the array contain non-NUL characters including the last character. The *version* field is two octets containing the characters `"00"` (zero-zero). The *typeflag* contains a single character. All other fields are leading zero-filled octal numbers using digits from the ISO/IEC 646:1991 standard IRV. Each numeric field is terminated by one or more <space> or NUL characters.

The *name* and the *prefix* fields shall produce the pathname of the file. A new pathname shall be formed, if *prefix* is not an empty string (its first character is not NUL), by concatenating *prefix* (up to the first NUL character), a <slash> character, and *name*; otherwise, *name* is used alone. In either case, *name* is terminated at the first NUL character. If *prefix* begins with a NUL character, it shall be ignored. In this manner, pathnames of at most 256 characters can be supported. If a pathname does not fit in the space provided, *pax* shall notify the user of the error, and shall not store any part of the file—header or data—on the medium.

The *linkname* field, described below, shall not use the *prefix* to produce a pathname. As such, a *linkname* is limited to 100 characters. If the name does not fit in the space provided, *pax* shall notify the user of the error, and shall not attempt to store the link on the medium.

The *mode* field provides 12 bits encoded in the ISO/IEC 646:1991 standard octal digit representation. The encoded bits shall represent the following values:

Table 3-16 ustar *mode* Field

Bit Value	POSIX.1-2024 Bit	Description
04 000	S_ISUID	Set UID on execution.
02 000	S_ISGID	Set GID on execution.
01 000	<reserved>	Reserved for future standardization.
00 400	S_IRUSR	Read permission for file owner class.
00 200	S_IWUSR	Write permission for file owner class.
00 100	S_IXUSR	Execute/search permission for file owner class.
00 040	S_IRGRP	Read permission for file group class.
00 020	S_IWGRP	Write permission for file group class.
00 010	S_IXGRP	Execute/search permission for file group class.
00 004	S_IROTH	Read permission for file other class.
00 002	S_IWOTH	Write permission for file other class.
00 001	S_IXOTH	Execute/search permission for file other class.

When appropriate privileges are required to set one of these mode bits, and the user restoring the files from the archive does not have appropriate privileges, the mode bits for which the user does not have appropriate privileges shall be ignored. Some of the mode bits in the archive format are not mentioned elsewhere in this volume of POSIX.1-2024. If the implementation does not support those bits, they may be ignored.

The *uid* and *gid* fields are the user and group ID of the owner and group of the file, respectively.

The *size* field is the size of the file in octets. If the *typeflag* field is set to specify a file to be of type 1 (a hard link) or 2 (a symbolic link), the *size* field shall be specified as zero. If the *typeflag* field is

set to specify a file of type 5 (directory), the *size* field shall be interpreted as described under the definition of that record type. No data logical records are stored for types 1, 2, or 5. If the *typeflag* field is set to 3 (character special file), 4 (block special file), or 6 (FIFO), the meaning of the *size* field is unspecified by this volume of POSIX.1-2024, and no data logical records shall be stored on the medium. Additionally, for type 6, the *size* field shall be ignored when reading. If the *typeflag* field is set to any other value, the number of logical records written following the header shall be (*size*+511)/512, ignoring any fraction in the result of the division.

The *mtime* field shall be the modification time of the file at the time it was archived. It is the ISO/IEC 646:1991 standard representation of the octal value of the modification time obtained from the *stat*( ) function.

The *chksum* field shall be the ISO/IEC 646:1991 standard IRV representation of the octal value of the simple sum of all octets in the header logical record. Each octet in the header shall be treated as an unsigned value. These values shall be added to an unsigned integer, initialized to zero, the precision of which is not less than 17 bits. When calculating the checksum, the *chksum* field is treated as if it were all <space> characters.

The *typeflag* field specifies the type of file archived. If a particular implementation does not recognize the type, or the user does not have appropriate privileges to create that type, the file shall be extracted as if it were a regular file if the file type is defined to have a meaning for the *size* field that could cause data logical records to be written on the medium (see the previous description for *size*). If conversion to a regular file occurs, the *pax* utility shall produce an error indicating that the conversion took place. All of the *typeflag* fields shall be coded in the ISO/IEC 646:1991 standard IRV:

`0` Represents a regular file. For backwards-compatibility, a *typeflag* value of binary zero (`'\0'`) should be recognized as meaning a regular file when extracting files from the archive. Archives written with this version of the archive file format create regular files with a *typeflag* value of the ISO/IEC 646:1991 standard IRV `'0'`.

`1` Represents a file linked to another file, of any type, previously archived. Such files are identified by having the same device and file serial numbers, and pathnames that refer to different directory entries. All such files shall be archived as linked files. The linked-to name is specified in the *linkname* field with a NUL-character terminator if it is less than 100 octets in length.

`2` Represents a symbolic link. The contents of the symbolic link shall be stored in the *linkname* field.

`3`, `4` Represent character special files and block special files respectively. In this case the *devmajor* and *devminor* fields shall contain information defining the device, the format of which is unspecified by this volume of POSIX.1-2024. Implementations may map the device specifications to their own local specification or may ignore the entry.

`5` Specifies a directory or subdirectory. On systems where disk allocation is performed on a directory basis, the *size* field shall contain the maximum number of octets (which may be rounded to the nearest disk block allocation unit) that the directory may hold. A *size* field of zero indicates no such limiting. Systems that do not support limiting in this manner should ignore the *size* field.

`6` Specifies a FIFO special file. Note that the archiving of a FIFO file archives the existence of this file and not its contents.

`7` Reserved to represent a file to which an implementation has associated some high-performance attribute. Implementations without such extensions should treat this file as a regular file (type 0).

110866 A–Z The letters 'A' to 'Z', inclusive, are reserved for custom implementations. All other
110867 values are reserved for future versions of this standard.

110868 It is unspecified whether files with pathnames that refer to the same directory entry are archived
110869 as linked files or as separate files. If they are archived as linked files, this means that attempting
110870 to extract both pathnames from the resulting archive always causes an error (unless the **–u**
110871 option is used) because the link cannot be created.

110872 It is unspecified whether files with the same device and file serial numbers being appended to
110873 an archive are treated as linked files to members that were in the archive before the append.

110874 Attempts to archive a socket shall produce a diagnostic message when **ustar** interchange format
110875 is used, but may be allowed when **pax** interchange format is used. Handling of other file types is
110876 implementation-defined.

110877 The *magic* field is the specification that this archive was output in this archive format. If this field
110878 contains **ustar** (the five characters from the ISO/IEC 646:1991 standard IRV shown followed by
110879 NUL), the *uname* and *gname* fields shall contain the ISO/IEC 646:1991 standard IRV
110880 representation of the owner and group of the file, respectively (truncated to fit, if necessary).
110881 When the file is restored by a privileged, protection-preserving version of the utility, the user
110882 and group databases shall be scanned for these names. If found, the user and group IDs
110883 contained within these files shall be used rather than the values contained within the *uid* and *gid*
110884 fields.

110885 **cpio Interchange Format**

110886 The octet-oriented **cpio** archive format shall be a series of entries, each comprising a header that
110887 describes the file, the name of the file, and then the contents of the file.

110888 An archive may be recorded as a series of fixed-size blocks of octets. This blocking shall be used
110889 only to make physical I/O more efficient. The last group of blocks shall always be at the full
110890 size.

110891 For the octet-oriented **cpio** archive format, the individual entry information shall be in the order
110892 indicated and described by the following table; see also the **<cpio.h>** header.

110893 **Table 3-17** Octet-Oriented cpio Archive Entry

110894 **Header Field Name**	**Length (in Octets)**	**Interpreted as**
110895 *c_magic*	6	Octal number
110896 *c_dev*	6	Octal number
110897 *c_ino*	6	Octal number
110898 *c_mode*	6	Octal number
110899 *c_uid*	6	Octal number
110900 *c_gid*	6	Octal number
110901 *c_nlink*	6	Octal number
110902 *c_rdev*	6	Octal number
110903 *c_mtime*	11	Octal number
110904 *c_namesize*	6	Octal number
110905 *c_filesize*	11	Octal number
110906 **Filename Field Name**	**Length**	**Interpreted as**
110907 *c_name*	*c_namesize*	Pathname string
110908 **File Data Field Name**	**Length**	**Interpreted as**
110909 *c_filedata*	*c_filesize*	Data

110910 **cpio Header**

110911 For each file in the archive, a header as defined previously shall be written. The information in
110912 the header fields is written as streams of the ISO/IEC 646:1991 standard characters interpreted
110913 as octal numbers. The octal numbers shall be extended to the necessary length by appending the
110914 ISO/IEC 646:1991 standard IRV zeros at the most-significant-digit end of the number; the result
110915 is written to the most-significant digit of the stream of octets first. The fields shall be interpreted
110916 as follows:

110917 *c_magic* Identify the archive as being a transportable archive by containing the identifying
110918 value "070707".

110919 *c_dev*, *c_ino* Contains values that uniquely identify the file within the archive (that is, no files
110920 contain the same pair of *c_dev* and *c_ino* values unless they are links to the same
110921 file). The values shall be determined in an unspecified manner.

110922 *c_mode* Contains the file type and access permissions as defined in the following table.

110923 **Table 3-18** Values for cpio c_mode Field

File Permissions Name	Value	Indicates
C_IRUSR	000 400	Read by owner
C_IWUSR	000 200	Write by owner
C_IXUSR	000 100	Execute by owner
C_IRGRP	000 040	Read by group
C_IWGRP	000 020	Write by group
C_IXGRP	000 010	Execute by group
C_IROTH	000 004	Read by others
C_IWOTH	000 002	Write by others
C_IXOTH	000 001	Execute by others
C_ISUID	004 000	Set *uid*
C_ISGID	002 000	Set *gid*
C_ISVTX	001 000	Reserved
File Type Name	**Value**	**Indicates**
C_ISDIR	040 000	Directory
C_ISFIFO	010 000	FIFO
C_ISREG	0100 000	Regular file
C_ISLNK	0120 000	Symbolic link
C_ISBLK	060 000	Block special file
C_ISCHR	020 000	Character special file
C_ISSOCK	0140 000	Socket
C_ISCTG	0110 000	Reserved

Directories, FIFOs, symbolic links, and regular files shall be supported on a system conforming to this volume of POSIX.1-2024; additional values defined previously are reserved for compatibility with existing systems. Additional file types may be supported; however, such files should not be written to archives intended to be transported to other systems.

c_uid Contains the user ID of the owner.

c_gid Contains the group ID of the group.

c_nlink Contains a number greater than or equal to the number of links in the archive referencing the file. If the **−a** option is used to append to a *cpio* archive, then the *pax* utility need not account for the files in the existing part of the archive when calculating the *c_nlink* values for the appended part of the archive, and need not alter the *c_nlink* values in the existing part of the archive if additional files with the same *c_dev* and *c_ino* values are appended to the archive.

c_rdev Contains implementation-defined information for character or block special files.

c_mtime Contains the latest time of modification of the file at the time the archive was created.

c_namesize Contains the length of the pathname, including the terminating NUL character.

c_filesize Contains the length in octets of the data section following the header structure.

110964 **cpio Filename**

110965 The *c_name* field shall contain the pathname of the file. The length of this field in octets is the
110966 value of *c_namesize*.

110967 If a filename is found on the medium that would create an invalid pathname, it is
110968 implementation-defined whether the data from the file is stored on the file hierarchy and under
110969 what name it is stored.

110970 All characters shall be represented in the ISO/IEC 646:1991 standard IRV. For maximum
110971 portability between implementations, names should be selected from characters represented by
110972 the portable filename character set as octets with the most significant bit zero. If an
110973 implementation supports the use of characters outside the portable filename character set in
110974 names for files, users, and groups, one or more implementation-defined encodings of these
110975 characters shall be provided for interchange purposes. However, the *pax* utility shall never create
110976 filenames on the local system that cannot be accessed via the procedures described previously in
110977 this volume of POSIX.1-2024. If a filename is found on the medium that would create an invalid
110978 filename, it is implementation-defined whether the data from the file is stored on the local file
110979 system and under what name it is stored. The *pax* utility may choose to ignore these files as long
110980 as it produces an error indicating that the file is being ignored.

110981 **cpio File Data**

110982 Following *c_name*, there shall be *c_filesize* octets of data. Interpretation of such data occurs in a
110983 manner dependent on the file. For regular files, the data shall consist of the contents of the file.
110984 For symbolic links, the data shall consist of the contents of the symbolic link. If *c_filesize* is zero,
110985 no data shall be contained in *c_filedata*.

110986 When restoring from an archive:

110987 • If the user does not have appropriate privileges to create a file of the specified type, *pax*
110988 shall ignore the entry and write an error message to standard error.

110989 • Only regular files and symbolic links have data to be restored. Presuming a regular file
110990 meets any selection criteria that might be imposed on the format-reading utility by the
110991 user, such data shall be restored.

110992 • If a user does not have appropriate privileges to set a particular mode flag, the flag shall be
110993 ignored. Some of the mode flags in the archive format are not mentioned elsewhere in this
110994 volume of POSIX.1-2024. If the implementation does not support those flags, they may be
110995 ignored.

110996 **cpio Special Entries**

110997 FIFO special files, directories, and the trailer shall be recorded with *c_filesize* equal to zero.
110998 Symbolic links shall be recorded with *c_filesize* equal to the length of the contents of the symbolic
110999 link. For other special files, *c_filesize* is unspecified by this volume of POSIX.1-2024. The header
111000 for the next file entry in the archive shall be written directly after the last octet of the file entry
111001 preceding it. A header denoting the filename **TRAILER!!!** shall indicate the end of the archive;
111002 the contents of octets in the last block of the archive following such a header are undefined.

111003 **EXIT STATUS**

111004 The following exit values shall be returned:

111005 0 All files were processed successfully.

111006 >0 An error occurred.

111007 **CONSEQUENCES OF ERRORS**

111008 If *pax* cannot create a file or a link when reading an archive or cannot find a file when writing an
111009 archive, or cannot preserve the user ID, group ID, or file mode when the **−p** option is specified, a
111010 diagnostic message shall be written to standard error and a non-zero exit status shall be
111011 returned, but processing shall continue. In the case where *pax* cannot create a hard link to a file,
111012 *pax* shall not, by default, create a second copy of the file.

111013 If the extraction of a file from an archive is prematurely terminated by a signal or error, *pax* may
111014 have only partially extracted the file or (if the **−n** option was not specified) may have extracted a
111015 file of the same name as that specified by the user, but which is not the file the user wanted.
111016 Additionally, the file modes of extracted directories may have additional bits from the S_IRWXU
111017 mask set as well as incorrect modification and access times.

111018 **APPLICATION USAGE**

111019 Caution is advised when using the **−a** option to append to a *cpio* format archive. If any of the
111020 files being appended happen to be given the same *c_dev* and *c_ino* values as a file in the existing
111021 part of the archive, then they may be treated as links to that file on extraction. Thus, it is risky to
111022 use **−a** with *cpio* format except when it is done on the same system that the original archive was
111023 created on, and with the same *pax* utility, and in the knowledge that there has been little or no
111024 file system activity since the original archive was created that could lead to any of the files
111025 appended being given the same *c_dev* and *c_ino* values as an unrelated file in the existing part of
111026 the archive. Also, when (intentionally) appending additional links to a file in the existing part of
111027 the archive, the *c_nlink* values in the modified archive can be smaller than the number of links to
111028 the file in the archive, which may mean that the links are not preserved on extraction.

111029 The **−p** (privileges) option was invented to reconcile differences between historical *tar* and *cpio*
111030 implementations. In particular, the two utilities use **−m** in diametrically opposed ways. The **−p**
111031 option also provides a consistent means of extending the ways in which future file attributes can
111032 be addressed, such as for enhanced security systems or high-performance files. Although it may
111033 seem complex, there are really two modes that are most commonly used:

111034 **−p e** ``Preserve everything''. This would be used by the historical superuser, someone with
111035 all appropriate privileges, to preserve all aspects of the files as they are recorded in the
111036 archive. The **e** flag is the sum of **o** and **p**, and other implementation-defined attributes.

111037 **−p p** ``Preserve'' the file mode bits. This would be used by the user with regular privileges
111038 who wished to preserve aspects of the file other than the ownership. The file times are
111039 preserved by default, but two other flags are offered to disable these and use the time
111040 of extraction.

111041 The one pathname per line format of standard input precludes pathnames containing **<newline>**
111042 characters. Although such pathnames violate the portable filename guidelines, they may exist
111043 and their presence may inhibit usage of *pax* within shell scripts. This problem is inherited from
111044 historical archive programs. The problem can be avoided by listing filename arguments on the
111045 command line instead of on standard input.

111046 It is almost certain that appropriate privileges are required for *pax* to accomplish parts of this
111047 volume of POSIX.1-2024. Specifically, creating files of type block special or character special,
111048 restoring file access times unless the files are owned by the user (the **−t** option), or preserving file
111049 owner, group, and mode (the **−p** option) all probably require appropriate privileges.

111050 In **read** mode, implementations are permitted to overwrite files when the archive has multiple
111051 members with the same name. This may fail if permissions on the first version of the file do not
111052 permit it to be overwritten.

The **cpio** and **ustar** formats can only support files up to 8 589 934 592 bytes (8 * 2^30) in size.

When archives containing binary header information are listed , the filenames printed may cause strange behavior on some terminals.

When all of the following are true:

1. A file of type directory is being placed into an archive.
2. The **ustar** archive format is being used.
3. The pathname of the directory is less than or equal to 155 bytes long (it will fit in the *prefix* field in the **ustar** header block).
4. The last component of the pathname of the directory is longer than 100 bytes long (it will not fit in the *name* field in the **ustar** header block).

some implementations of the *pax* utility will place the entire directory pathname in the *prefix* field, set the *name* field to an empty string, and place the directory in the archive. Other implementations of the *pax* utility will give an error under these conditions because the *name* field is not large enough to hold the last component of the directory name. This standard allows either behavior. However, when extracting a directory from a **ustar** format archive, this standard requires that all implementations be able to extract a directory even if the *name* field contains an empty string as long as the *prefix* field does not also contain an empty string.

When restricting file hierarchy traversal to one file system, it can sometimes be desirable for the crossing points themselves to be processed (archived or copied) and sometimes for them not to be processed. (Crossing points are mount points and, if the **–L** option is specified, symbolic links to directories on other file systems.) With the **–X** option *pax* processes them, but there is no standard way to have *pax* not process them. However, this can be achieved by using *find* to do the hierarchy traversal and piping the output of find to *pax* (with the **–d** option); see the APPLICATION USAGE for *find*.

EXAMPLES

The following command:

```
pax –w –f /dev/rmt/1m .
```

copies the contents of the current directory to tape drive 1, medium density (assuming historical System V device naming procedures—the historical BSD device name would be **/dev/rmt9**).

The following commands:

```
mkdir newdir
pax –rw olddir newdir
```

copy the *olddir* directory hierarchy to *newdir*.

```
pax –r –s ',^//*usr//*,,' –f a.pax
```

reads the archive **a.pax**, with all files rooted in **/usr** in the archive extracted relative to the current directory.

Using the option:

```
–o listopt="%M %(atime)T %(size)D %(name)s"
```

overrides the default output description in Standard Output and instead writes:

```
–rw–rw––– Jan 12 15:53 2003 1492 /usr/foo/bar
```

Using the options:


```
-o listopt='%L\t%(size)D\n%.7' \
-o listopt='(name)s\n%(atime)T\n%T'
```

overrides the default output description in Standard Output and instead writes:

```
/usr/foo/bar -> /tmp   1492
/usr/fo
Jan 12 15:53 1991
Jan 31 15:53 2003
```

RATIONALE

The *pax* utility was new for the ISO POSIX-2: 1993 standard. It represents a peaceful compromise between advocates of the historical *tar* and *cpio* utilities.

A fundamental difference between *cpio* and *tar* was in the way directories were treated. The *cpio* utility did not treat directories differently from other files, and to select a directory and its contents required that each file in the hierarchy be explicitly specified. For *tar*, a directory matched every file in the file hierarchy it rooted.

The *pax* utility offers both interfaces; by default, directories map into the file hierarchy they root. The **–d** option causes *pax* to skip any file not explicitly referenced, as *cpio* historically did. The *tar* *–style* behavior was chosen as the default because it was believed that this was the more common usage and because *tar* is the more commonly available interface, as it was historically provided on both System V and BSD implementations.

The data interchange format specification in this volume of POSIX.1-2024 requires that processes with ``appropriate privileges'' shall always restore the ownership and permissions of extracted files exactly as archived. If viewed from the historic equivalence between superuser and ``appropriate privileges'', there are two problems with this requirement. First, users running as superusers may unknowingly set dangerous permissions on extracted files. Second, it is needlessly limiting, in that superusers cannot extract files and own them as superuser unless the archive was created by the superuser. (It should be noted that restoration of ownerships and permissions for the superuser, by default, is historical practice in *cpio*, but not in *tar*.) In order to avoid these two problems, the *pax* specification has an additional ``privilege'' mechanism, the **–p** option. Only a *pax* invocation with the privileges needed, and which has the **–p** option set using the `e` specification character, has appropriate privileges to restore full ownership and permission information.

Note also that this volume of POSIX.1-2024 requires that the file ownership and access permissions shall be set, on extraction, in the same fashion as the *creat*( ) function when provided with the mode stored in the archive. This means that the file creation mask of the user is applied to the file permissions.

Users should note that directories may be created by *pax* while extracting files with permissions that are different from those that existed at the time the archive was created. When extracting sensitive information into a directory hierarchy that no longer exists, users are encouraged to set their file creation mask appropriately to protect these files during extraction.

The table of contents output is written to standard output to facilitate pipeline processing.

An early proposal had hard links displaying for all pathnames. This was removed because it complicates the output of the case where **–v** is not specified and does not match historical *cpio* usage. The hard-link information is available in the **–v** display.

The description of the **–l** option allows implementations to make hard links to symbolic links. Earlier versions of this standard did not specify any way to create a hard link to a symbolic link, but many implementations provided this capability as an extension. If there are hard links to

111140 symbolic links when an archive is created, the implementation is required to archive the hard
111141 link in the archive (unless **−H** or **−L** is specified). When in **read** mode and in **copy** mode,
111142 implementations supporting hard links to symbolic links should use them when appropriate.

111143 The archive formats inherited from the POSIX.1-1990 standard have certain restrictions that have
111144 been brought along from historical usage. For example, there are restrictions on the length of
111145 pathnames stored in the archive. When *pax* is used in **copy**(**−rw**) mode (copying directory
111146 hierarchies), the ability to use extensions from the **−xpax** format overcomes these restrictions.

111147 The default *blocksize* value of 5 120 bytes for *cpio* was selected because it is one of the standard
111148 block-size values for *cpio*, set when the **−B** option is specified. (The other default block-size value
111149 for *cpio* is 512 bytes, and this was considered to be too small.) The default block value of 10 240
111150 bytes for *tar* was selected because that is the standard block-size value for BSD *tar*. The
111151 maximum block size of 32 256 bytes (2^{15}–512 bytes) is the largest multiple of 512 bytes that fits
111152 into a signed 16-bit tape controller transfer register. There are known limitations in some
111153 historical systems that would prevent larger blocks from being accepted. Historical values were
111154 chosen to improve compatibility with historical scripts using *dd* or similar utilities to manipulate
111155 archives. Also, default block sizes for any file type other than character special file has been
111156 deleted from this volume of POSIX.1-2024 as unimportant and not likely to affect the structure of
111157 the resulting archive.

111158 Implementations are permitted to modify the block-size value based on the archive format or the
111159 device to which the archive is being written. This is to provide implementations with the
111160 opportunity to take advantage of special types of devices, and it should not be used without a
111161 great deal of consideration as it almost certainly decreases archive portability.

111162 The intended use of the **−n** option was to permit extraction of one or more files from the archive
111163 without processing the entire archive. This was viewed by the standard developers as offering
111164 significant performance advantages over historical implementations. The **−n** option in early
111165 proposals had three effects; the first was to cause special characters in patterns to not be treated
111166 specially. The second was to cause only the first file that matched a pattern to be extracted. The
111167 third was to cause *pax* to write a diagnostic message to standard error when no file was found
111168 matching a specified pattern. Only the second behavior is retained by this volume of
111169 POSIX.1-2024, for many reasons. First, it is in general not acceptable for a single option to have
111170 multiple effects. Second, the ability to make pattern matching characters act as normal characters
111171 is useful for parts of *pax* other than file extraction. Third, a finer degree of control over the
111172 special characters is useful because users may wish to normalize only a single special character
111173 in a single filename. Fourth, given a more general escape mechanism, the previous behavior of
111174 the **−n** option can be easily obtained using the **−s** option or a *sed* script. Finally, writing a
111175 diagnostic message when a pattern specified by the user is unmatched by any file is useful
111176 behavior in all cases.

111177 In this version, the **−n** was removed from the **copy** mode synopsis of *pax*; it is inapplicable
111178 because there are no pattern operands specified in this mode.

111179 There is another method than *pax* for copying subtrees in POSIX.1-2024 described as part of the
111180 *cp* utility. Both methods are historical practice: *cp* provides a simpler, more intuitive interface,
111181 while *pax* offers a finer granularity of control. Each provides additional functionality to the
111182 other; in particular, *pax* maintains the hard-link structure of the hierarchy while *cp* does not. It is
111183 the intention of the standard developers that the results be similar (using appropriate option
111184 combinations in both utilities). The results are not required to be identical; there seemed
111185 insufficient gain to applications to balance the difficulty of implementations having to guarantee
111186 that the results would be exactly identical.

111187 A single archive may span more than one file. It is suggested that implementations provide

111188 informative messages to the user on standard error whenever the archive file is changed.

111189 The –**d** option (do not create intermediate directories not listed in the archive) found in early
111190 proposals was originally provided as a complement to the historic –**d** option of *cpio*. It has been
111191 deleted.

111192 The –**s** option in early proposals specified a subset of the substitution command from the *ed*
111193 utility. As there was no reason for only a subset to be supported, the –**s** option is now compatible
111194 with the current *ed* specification. Since the delimiter can be any non-null character, the following
111195 usage with single <space> characters is valid:

111196 `pax -s " foo bar " ...`

111197 The –**t** description is worded so as to note that this may cause the access time update caused by
111198 some other activity (which occurs while the file is being read) to be overwritten.

111199 The default behavior of *pax* with regard to file modification times is the same as historical
111200 implementations of *tar*. It is not the historical behavior of *cpio*.

111201 Because the –**i** option uses **/dev/tty**, utilities without a controlling terminal are not able to use
111202 this option.

111203 The –**y** option, found in early proposals, has been deleted because a line containing a single
111204 <period> for the –**i** option has equivalent functionality. The special lines for the –**i** option (a
111205 single <period> and the empty line) are historical practice in *cpio*.

111206 In early drafts, a –**e***charmap* option was included to increase portability of files between systems
111207 using different coded character sets. This option was omitted because it was apparent that
111208 consensus could not be formed for it. In this version, the use of UTF-8 should be an adequate
111209 substitute.

111210 The ISO POSIX-2:1993 standard and ISO POSIX-1 standard requirements for *pax*, however,
111211 made it very difficult to create a single archive containing files created using extended characters
111212 provided by different locales. This version adds the **hdrcharset** keyword to make it possible to
111213 archive files in these cases without dropping files due to translation errors.

111214 Translating filenames and other attributes from a locale's encoding to UTF-8 and then back again
111215 can lose information, as the resulting filename might not be byte-for-byte equivalent to the
111216 original. To avoid this problem, users can specify the –**o hdrcharset=binary** option, which will
111217 cause the resulting archive to use binary format for all names and attributes. Such archives are
111218 not portable among hosts that use different native encodings (e.g., EBCDIC *versus* ASCII-based
111219 encodings), but they will allow interchange among the vast majority of POSIX file systems in
111220 practical use. Also, the –**o hdrcharset=binary** option will cause *pax* in **copy** mode to behave
111221 more like other standard utilities such as *cp*.

111222 If the values specified by the –**o exthdr.name=value**, –**o globexthdr.name=value**, or by
111223 **$TMPDIR** (if –**o globexthdr.name** is not specified) require a character encoding other than that
111224 described in the ISO/IEC 646:1991 standard, a **path** extended header record will have to be
111225 created for the file. If a **hdrcharset** extended header record is active for such headers, it will
111226 determine the codeset used for the value field in these extended **path** header records. These **path**
111227 extended header records always need to be created when writing an archive even if
111228 **hdrcharset=binary** has been specified and would contain the same (binary) data that appears in
111229 the **ustar** header record prefix and *name* fields. (In other words, an extended header **path** record
111230 is always required to be generated if the *prefix* or *name* fields contain non-ASCII characters even
111231 when **hdrcharset=binary** is also in effect for that file.)

111232 The –**k** option was added to address international concerns about the dangers involved in the
111233 character set transformations of –**e** (if the target character set were different from the source, the

filenames might be transformed into names matching existing files) and also was made more general to protect files transferred between file systems with different {NAME_MAX} values (truncating a filename on a smaller system might also inadvertently overwrite existing files). As stated, it prevents any overwriting, even if the target file is older than the source. This version adds more granularity of options to solve this problem by introducing the –**oinvalid=option**—specifically the **UTF-8** and **binary** actions. (Note that an existing file is still subject to overwriting in this case. The –**k** option closes that loophole.)

Some of the file characteristics referenced in this volume of POSIX.1-2024 might not be supported by some archive formats. For example, neither the **tar** nor **cpio** formats contain the file access time. For this reason, the `e` specification character has been provided, intended to cause all file characteristics specified in the archive to be retained.

It is required that extracted directories, by default, have their access and modification times and permissions set to the values specified in the archive. This has obvious problems in that the directories are almost certainly modified after being extracted and that directory permissions may not permit file creation. One possible solution is to create directories with the mode specified in the archive, as modified by the *umask* of the user, with sufficient permissions to allow file creation. After all files have been extracted, *pax* would then reset the access and modification times and permissions as necessary.

The list-mode formatting description borrows heavily from the one defined by the *printf* utility. However, since there is no separate operand list to get conversion arguments, the format was extended to allow specifying the name of the conversion argument as part of the conversion specification.

The `T` conversion specifier allows time fields to be displayed in any of the date formats. Unlike the *ls* utility, *pax* does not adjust the format when the date is less than six months in the past. This makes parsing the output more predictable.

The `D` conversion specifier handles the ability to display the major/minor or file size, as with *ls*, by using `%–8(`*`size`*`)D`.

The `L` conversion specifier handles the *ls* display for symbolic links.

Conversion specifiers were added to generate existing known types used for *ls*.

pax Interchange Format

The new POSIX data interchange format was developed primarily to satisfy international concerns that the **ustar** and **cpio** formats did not provide for file, user, and group names encoded in characters outside a subset of the ISO/IEC 646:1991 standard. The standard developers realized that this new POSIX data interchange format should be very extensible because there were other requirements they foresaw in the near future:

- Support international character encodings and locale information
- Support security information (ACLs, and so on)
- Support future file types, such as realtime or contiguous files
- Include data areas for implementation use
- Support systems with words larger than 32 bits and timers with subsecond granularity

The following were not goals for this format because these are better handled by separate utilities or are inappropriate for a portable format:

- Encryption
- Compression
- Data translation between locales and codesets
- *inode* storage

The format chosen to support the goals is an extension of the **ustar** format. Of the two formats previously available, only the **ustar** format was selected for extensions because:

- It was easier to extend in an upwards-compatible way. It offered version flags and header block type fields with room for future standardization. The **cpio** format, while possessing a more flexible file naming methodology, could not be extended without breaking some theoretical implementation or using a dummy filename that could be a legitimate filename.
- Industry experience since the original ``*tar* wars'' fought in developing the ISO POSIX-1 standard has clearly been in favor of the **ustar** format, which is generally the default output format selected for *pax* implementations on new systems.

The new format was designed with one additional goal in mind: reasonable behavior when an older *tar* or *pax* utility happened to read an archive. Since the POSIX.1-1990 standard mandated that a ``format-reading utility'' had to treat unrecognized *typeflag* values as regular files, this allowed the format to include all the extended information in a pseudo-regular file that preceded each real file. An option is given that allows the archive creator to set up reasonable names for these files on the older systems. Also, the normative text suggests that reasonable file access values be used for this **ustar** header block. Making these header files inaccessible for convenient reading and deleting would not be reasonable. File permissions of 600 or 700 are suggested.

The **ustar** *typeflag* field was used to accommodate the additional functionality of the new format rather than magic or version because the POSIX.1-1990 standard (and, by reference, the previous version of *pax*), mandated the behavior of the format-reading utility when it encountered an unknown *typeflag*, but was silent about the other two fields.

Early proposals for the first version of this standard contained a proposed archive format that was based on compatibility with the standard for tape files (ISO 1001, similar to the format used historically on many mainframes and minicomputers). This format was overly complex and required considerable overhead in volume and header records. Furthermore, the standard developers felt that it would not be acceptable to the community of POSIX developers, so it was later changed to be a format more closely related to historical practice on POSIX systems.

The prefix and name split of pathnames in **ustar** was replaced by the single path extended header record for simplicity.

The concept of a global extended header (*typeflag* **g**) was controversial. If this were applied to an archive being recorded on magnetic tape, a few unreadable blocks at the beginning of the tape could be a serious problem; a utility attempting to extract as many files as possible from a damaged archive could lose a large percentage of file header information in this case. However, if the archive were on a reliable medium, such as a CD-ROM, the global extended header offers considerable potential size reductions by eliminating redundant information. Thus, the text warns against using the global method for unreliable media and provides a method for implanting global information in the extended header for each file, rather than in the *typeflag* **g** records.

No facility for data translation or filtering on a per-file basis is included because the standard developers could not invent an interface that would allow this in an efficient manner. If a filter, such as encryption or compression, is to be applied to all the files, it is more efficient to apply the

filter to the entire archive as a single file. The standard developers considered interfaces that would invoke a shell script for each file going into or out of the archive, but the system overhead in this approach was considered to be too high.

One such approach would be to have **filter=** records that give a pathname for an executable. When the program is invoked, the file and archive would be open for standard input/output and all the header fields would be available as environment variables or command-line arguments. The standard developers did discuss such schemes, but they were omitted from POSIX.1-2024 due to concerns about excessive overhead. Also, the program itself would need to be in the archive if it were to be used portably.

There is currently no portable means of identifying the character set(s) used for a file in the file system. Therefore, *pax* has not been given a mechanism to generate charset records automatically. The only portable means of doing this is for the user to write the archive using the −**ocharset=string** command line option. This assumes that all of the files in the archive use the same encoding. The ``implementation-defined'' text is included to allow for a system that can identify the encodings used for each of its files.

The table of standards that accompanies the charset record description is acknowledged to be very limited. Only a limited number of character set standards is reasonable for maximal interchange. Any character set is, of course, possible by prior agreement. It was suggested that EBCDIC be listed, but it was omitted because it is not defined by a formal standard. Formal standards, and then only those with reasonably large followings, can be included here, simply as a matter of practicality. The <*value*>s represent names of officially registered character sets in the format required by the ISO 2375: 1985 standard.

The normal <comma> or <blank>-separated list rules are not followed in the case of keyword options to allow ease of argument parsing for *getopts*.

Further information on character encodings is in pax Archive Character Set Encoding/Decoding (on page 3284).

The standard developers have reserved keyword name space for vendor extensions. It is suggested that the format to be used is:

```
VENDOR.keyword
```

where *VENDOR* is the name of the vendor or organization in all uppercase letters. It is further suggested that the keyword following the <period> be named differently than any of the standard keywords so that it could be used for future standardization, if appropriate, by omitting the *VENDOR* prefix.

The <*length*> field in the extended header record was included to make it simpler to step through the records, even if a record contains an unknown format (to a particular *pax*) with complex interactions of special characters. It also provides a minor integrity checkpoint within the records to aid a program attempting to recover files from a damaged archive.

There are no extended header versions of the *devmajor* and *devminor* fields because the unspecified format **ustar** header field should be sufficient. If they are not, vendor-specific extended keywords (such as *VENDOR.devmajor*) should be used.

Device and *i*-number labeling of files was not adopted from *cpio*; files are interchanged strictly on a symbolic name basis, as in **ustar**.

Just as with the **ustar** format descriptions, the new format makes no special arrangements for multi-volume archives. Each of the *pax* archive types is assumed to be inside a single POSIX file and splitting that file over multiple volumes (diskettes, tape cartridges, and so on), processing their labels, and mounting each in the proper sequence are considered to be implementation

111368 details that cannot be described portably.

111369 The **pax** format is intended for interchange, not only for backup on a single (family of) systems.
111370 It is not as densely packed as might be possible for backup:

111371 • It contains information as coded characters that could be coded in binary.

111372 • It identifies extended records with name fields that could be omitted in favor of a fixed-
111373 field layout.

111374 • It translates names into a portable character set and identifies locale-related information,
111375 both of which are probably unnecessary for backup.

111376 The requirements on restoring from an archive are slightly different from the historical wording,
111377 allowing for non-monolithic privilege to bring forward as much as possible. In particular,
111378 attributes such as ``high performance file'' might be broadly but not universally granted while
111379 set-user-ID or *chown*() might be much more restricted. There is no implication in POSIX.1-2024
111380 that the security information be honored after it is restored to the file hierarchy, in spite of what
111381 might be improperly inferred by the silence on that topic. That is a topic for another standard.

111382 Hard links are recorded in the fashion described here because a hard link can be to any file type.
111383 It is desirable in general to be able to restore part of an archive selectively and restore all of those
111384 files completely. If the data is not associated with each hard link, it is not possible to do this.
111385 However, the data associated with a file can be large, and when selective restoration is not
111386 needed, this can be a significant burden. The archive is structured so that files that have no
111387 associated data can always be restored by the name of any link name of any hard link, and the
111388 user can choose whether data is recorded with each instance of a file that contains data. The
111389 format permits mixing of hard links with data and hard links without data in a single archive;
111390 this can be done for special needs, and *pax* is expected to interpret such archives on input
111391 properly, despite the fact that there is no *pax* option that would force this mixed case on output.
111392 (When **−o linkdata** is used, the output must contain the duplicate data, but the implementation
111393 is free to include it or omit it when **−o linkdata** is not used.)

111394 The time values are included as extended header records for those implementations needing
111395 more than the eleven octal digits allowed by the **ustar** format. Portable file timestamps cannot be
111396 negative. If *pax* encounters a file with a negative timestamp in **copy** or **write** mode, it can reject
111397 the file, substitute a non-negative timestamp, or generate a non-portable timestamp with a
111398 leading `'-'`. Even though some implementations can support finer file-time granularities than
111399 seconds, the normative text requires support only for seconds since the Epoch because the
111400 ISO POSIX-1 standard states them that way. The **ustar** format includes only *mtime*; the new
111401 format adds *atime* and *ctime* for symmetry. The *atime* access time restored to the file system will
111402 be affected by the **−p a** and **−p e** options. The *ctime* creation time (actually *inode* modification
111403 time) is described with appropriate privileges so that it can be ignored when writing to the file
111404 system. POSIX does not provide a portable means to change file creation time. Nothing is
111405 intended to prevent a non-portable implementation of *pax* from restoring the value.

111406 The *gid*, *size*, and *uid* extended header records were included to allow expansion beyond the
111407 sizes specified in the regular *tar* header. New file system architectures are emerging that will
111408 exhaust the 12-digit size field. There are probably not many systems requiring more than 8 digits
111409 for user and group IDs, but the extended header values were included for completeness,
111410 allowing overrides for all of the decimal values in the *tar* header.

111411 The standard developers intended to describe the effective results of *pax* with regard to file
111412 ownerships and permissions; implementations are not restricted in timing or sequencing the
111413 restoration of such, provided the results are as specified.

111414 Much of the text describing the extended headers refers to use in ``**write** or **copy** modes''. The

copy mode references are due to the normative text: ``The effect of the copy shall be as if the copied files were written to an archive file and then subsequently extracted ...''. There is certainly no way to test whether *pax* is actually generating the extended headers in **copy** mode, but the effects must be as if it had.

pax Archive Character Set Encoding/Decoding

There is a need to exchange archives of files between systems of different native codesets. Filenames, group names, and user names must be preserved to the fullest extent possible when an archive is read on the receiving platform. Translation of the contents of files is not within the scope of the *pax* utility.

There will also be the need to represent characters that are not available on the receiving platform. These unsupported characters cannot be automatically folded to the local set of characters due to the chance of collisions. This could result in overwriting previous extracted files from the archive or pre-existing files on the system.

For these reasons, the codeset used to represent characters within the extended header records of the *pax* archive must be sufficiently rich to handle all commonly used character sets. The fields requiring translation include, at a minimum, filenames, user names, group names, and link pathnames. Implementations may wish to have localized extended keywords that use non-portable characters.

The standard developers considered the following options:

- The archive creator specifies the well-defined name of the source codeset. The receiver must then recognize the codeset name and perform the appropriate translations to the destination codeset.
- The archive creator includes within the archive the character mapping table for the source codeset used to encode extended header records. The receiver must then read the character mapping table and perform the appropriate translations to the destination codeset.
- The archive creator translates the extended header records in the source codeset into a canonical form. The receiver must then perform the appropriate translations to the destination codeset.

The approach that incorporates the name of the source codeset poses the problem of codeset name registration, and makes the archive useless to *pax* archive decoders that do not recognize that codeset.

Because parts of an archive may be corrupted, the standard developers felt that including the character map of the source codeset was too fragile. The loss of this one key component could result in making the entire archive useless. (The difference between this and the global extended header decision was that the latter has a workaround—duplicating extended header records on unreliable media—but this would be too burdensome for large character set maps.)

Both of the above approaches also put an undue burden on the *pax* archive receiver to handle the cross-product of all source and destination codesets.

To simplify the translation from the source codeset to the canonical form and from the canonical form to the destination codeset, the standard developers decided that the internal representation should be a stateless encoding. A stateless encoding is one where each codepoint has the same meaning, without regard to the decoder being in a specific state. An example of a stateful encoding would be the Japanese Shift-JIS; an example of a stateless encoding would be the ISO/IEC 646: 1991 standard (equivalent to 7-bit ASCII).

For these reasons, the standard developers decided to adopt a canonical format for the representation of file information strings. The obvious, well-endorsed candidate is the ISO/IEC 10646:2020 standard (based in part on Unicode), which can be used to represent the characters of virtually all standardized character sets. The standard developers initially agreed upon using UCS2 (16-bit Unicode) as the internal representation. This repertoire of characters provides a sufficiently rich set to represent all commonly-used codesets.

However, the standard developers found that the 16-bit Unicode representation had some problems. It forced the issue of standardizing byte ordering. The 2-byte length of each character made the extended header records twice as long for the case of strings coded entirely from historical 7-bit ASCII. For these reasons, the standard developers chose the UTF-8 defined in the ISO/IEC 10646:2020 standard. This multi-byte representation encodes UCS2 or UCS4 characters reliably and deterministically, eliminating the need for a canonical byte ordering. In addition, NUL octets and other characters possibly confusing to POSIX file systems do not appear, except to represent themselves. It was realized that certain national codesets take up more space after the encoding, due to their placement within the UCS range; it was felt that the usefulness of the encoding of the names outweighs the disadvantage of size increase for file, user, and group names.

The encoding of UTF-8 is as follows:

```
UCS4 Hex Encoding  UTF-8 Binary Encoding

00000000-0000007F  0xxxxxxx
00000080-000007FF  110xxxxx 10xxxxxx
00000800-0000FFFF  1110xxxx 10xxxxxx 10xxxxxx
00010000-001FFFFF  11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
00200000-03FFFFFF  111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
04000000-7FFFFFFF  1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
```

where each `'x'` represents a bit value from the character being translated.

ustar Interchange Format

The description of the **ustar** format reflects numerous enhancements over pre-1988 versions of the historical *tar* utility. The goal of these changes was not only to provide the functional enhancements desired, but also to retain compatibility between new and old versions. This compatibility has been retained. Archives written using the old archive format are compatible with the new format.

Implementors should be aware that the previous file format did not include a mechanism to archive directory type files. For this reason, the convention of using a filename ending with <slash> was adopted to specify a directory on the archive.

The total size of the *name* and *prefix* fields have been set to meet the minimum requirements for {PATH_MAX}. If a pathname will fit within the *name* field, it is recommended that the pathname be stored there without the use of the *prefix* field. Although the name field is known to be too small to contain {PATH_MAX} characters, the value was not changed in this version of the archive file format to retain backwards-compatibility, and instead the prefix was introduced. Also, because of the earlier version of the format, there is no way to remove the restriction on the *linkname* field being limited in size to just that of the *name* field.

The *size* field is required to be meaningful in all implementation extensions, although it could be zero. This is required so that the data blocks can always be properly counted.

It is suggested that if device special files need to be represented that cannot be represented in the standard format, that one of the extension types (**A**-**Z**) be used, and that the additional

111506 information for the special file be represented as data and be reflected in the *size* field.

111507 Attempting to restore a special file type, where it is converted to ordinary data and conflicts with
111508 an existing filename, need not be specially detected by the utility. If run as an ordinary user, *pax*
111509 should not be able to overwrite the entries in, for example, **/dev** in any case (whether the file is
111510 converted to another type or not). If run as a privileged user, it should be able to do so, and it
111511 would be considered a bug if it did not. The same is true of ordinary data files and similarly
111512 named special files; it is impossible to anticipate the needs of the user (who could really intend
111513 to overwrite the file), so the behavior should be predictable (and thus regular) and rely on the
111514 protection system as required.

111515 The value 7 in the *typeflag* field is intended to define how contiguous files can be stored in a
111516 **ustar** archive. POSIX.1-2024 does not require the contiguous file extension, but does define a
111517 standard way of archiving such files so that all conforming systems can interpret these file types
111518 in a meaningful and consistent manner. On a system that does not support extended file types,
111519 the *pax* utility should do the best it can with the file and go on to the next.

111520 The file protection modes are those conventionally used by the *ls* utility. This is extended beyond
111521 the usage in the ISO POSIX-2 standard to support the ``shared text'' or ``sticky'' bit. It is intended
111522 that the conformance document should not document anything beyond the existence of and
111523 support of such a mode. Further extensions are expected to these bits, particularly with
111524 overloading the set-user-ID and set-group-ID flags.

111525 **cpio Interchange Format**

111526 The reference to appropriate privileges in the **cpio** format refers to an error on standard output;
111527 the **ustar** format does not make comparable statements.

111528 The model for this format was the historical System V *cpio*–**c** data interchange format. This
111529 model documents the portable version of the **cpio** format and not the binary version. It has the
111530 flexibility to transfer data of any type described within POSIX.1-2024, yet is extensible to transfer
111531 data types specific to extensions beyond POSIX.1-2024 (for example, contiguous files). Because it
111532 describes existing practice, there is no question of maintaining upwards-compatibility.

111533 **cpio Header**

111534 There has been some concern that the size of the *c_ino* field of the header is too small to handle
111535 those systems that have very large *inode* numbers. However, the *c_ino* field in the header is used
111536 strictly as a hard-link resolution mechanism for archives. It is not necessarily the same value as
111537 the *inode* number of the file in the location from which that file is extracted.

111538 The name *c_magic* is based on historical usage.

111539 **cpio Filename**

111540 For most historical implementations of the *cpio* utility, {PATH_MAX} octets can be used to
111541 describe the pathname without the addition of any other header fields (the NUL character
111542 would be included in this count). {PATH_MAX} is the minimum value for pathname size,
111543 documented as 256 bytes. However, an implementation may use *c_namesize* to determine the
111544 exact length of the pathname. With the current description of the **<cpio.h>** header, this
111545 pathname size can be as large as a number that is described in six octal digits.

111546 Two values are documented under the *c_mode* field values to provide for extensibility for known
111547 file types:

111548 **0110 000** Reserved for contiguous files. The implementation may treat the rest of the
111549 information for this archive like a regular file. If this file type is undefined, the
111550 implementation may create the file as a regular file.

111551 This provides for extensibility of the **cpio** format while allowing for the ability to read old
111552 archives. Files of an unknown type may be read as ``regular files'' on some implementations. On
111553 a system that does not support extended file types, the *pax* utility should do the best it can with
111554 the file and go on to the next.

111555 **FUTURE DIRECTIONS**

111556 If this utility is directed to display a pathname that contains any bytes that have the encoded
111557 value of a <newline> character when <newline> is a terminator or separator in the output
111558 format being used, implementations are encouraged to treat this as an error. A future version of
111559 this standard may require implementations to treat this as an error.

111560 If this utility is directed to create a new directory entry that contains any bytes that have the
111561 encoded value of a <newline> character, implementations are encouraged to treat this as an
111562 error. A future version of this standard may require implementations to treat this as an error.

111563 **SEE ALSO**

111564 Chapter 2 (on page 2472), *cp*, *ed*, *getopts*, *ls*, *printf*

111565 XBD Section 3.145 (on page 52), Chapter 5 (on page 113), Chapter 8 (on page 167), Section 12.2
111566 (on page 215), **<cpio.h>**, **<tar.h>**

111567 XSH *chown*( ), *creat*( ), *fstatat*( ), *futimens*( ), *mkdir*( ), *mkfifo*( ), *write*( )

111568 **CHANGE HISTORY**

111569 First released in Issue 4.

111570 **Issue 5**

111571 A note is added to the APPLICATION USAGE indicating that the **cpio** and **tar** formats can only
111572 support files up to 8 gigabytes in size.

111573 **Issue 6**

111574 The *pax* utility is aligned with the IEEE P1003.2b draft standard:

111575 • Support has been added for symbolic links in the options and interchange formats.

111576 • A new format has been devised, based on extensions to **ustar**.

111577 • References to the ``extended'' **tar** and **cpio** formats derived from the POSIX.1-1990
111578 standard have been changed to remove the ``extended'' adjective because this could cause
111579 confusion with the extended **tar** header added in this version. (All references to **tar** are
111580 actually to **ustar**.)

111581 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

111582 IEEE PASC Interpretation 1003.2 #168 is applied, clarifying that *mkdir*( ) and *mkfifo*( ) calls can
111583 ignore an [EEXIST] error when extracting an archive.

111584 IEEE PASC Interpretation 1003.2 #180 is applied, clarifying how extracted files are created when
111585 in **read** mode.

111586 IEEE PASC Interpretation 1003.2 #181 is applied, clarifying the description of the **–t** option.

111587 IEEE PASC Interpretation 1003.2 #195 is applied.

111588 IEEE PASC Interpretation 1003.2 #206 is applied, clarifying the handling of links for the **–H**, **–L**,
111589 and **–l** options.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/35 is applied, adding the process ID of the *pax* process into certain fields. This change provides a method for the implementation to ensure that different instances of *pax* extracting a file named **/a/b/foo** will not collide when processing the extended header information associated with **foo**.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/36 is applied, changing **–x B** to **–x** *pax* in the OPTIONS section.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/20 is applied, updating the SYNOPSIS to be consistent with the normative text.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/21 is applied, updating the DESCRIPTION to describe the behavior when files to be linked are symbolic links and the system is not capable of making hard links to symbolic links.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/22 is applied, updating the OPTIONS section to describe the behavior for how multiple **–odelete=pattern** options are to be handled.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/23 is applied, updating the **write** option within the OPTIONS section.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/24 is applied, adding a paragraph into the OPTIONS section that states that specifying more than one of the mutually-exclusive options (**–H** and **–L**) is not considered an error and that the last option specified will determine the behavior of the utility.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/25 is applied, removing the *ctime* paragraph within the EXTENDED DESCRIPTION. There is a contradiction in the definition of the *ctime* keyword for the *pax* extended header, in that the *st_ctime* member of the **stat** structure does not refer to a file creation time. No field in the standard **stat** structure from **<sys/stat.h>** includes a file creation time.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/26 is applied, making it clear that *typeflag* 1 (**ustar** Interchange Format) applies not only to files that are hard-linked, but also to files that are aliased via symbolic links.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/27 is applied, clarifying the *cpio c_nlink* field.

Issue 7

Austin Group Interpretations 1003.1-2001 #011, #036, #086, and #109 are applied.

Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the *LC_MESSAGES* environment variable.

SD5-XCU-ERN-2 is applied, making **–c** and **–n** mutually-exclusive in the SYNOPSIS.

SD5-XCU-ERN-3 is applied, revising the default behavior of **–H** and **–L**.

SD5-XCU-ERN-5, SD5-XCU-ERN-6, SD5-XCU-ERN-7, SD5-XCU-ERN-60 are applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The *pax* utility is no longer allowed to create separate identical symbolic links when extracting linked symbolic links from an archive.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0128 [260], XCU/TC1-2008/0129 [261], XCU/TC1-2008/0130 [261], XCU/TC1-2008/0131 [313], and XCU/TC1-2008/0132 [233] are applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0152 [886], XCU/TC2-2008/0153

[814], XCU/TC2-2008/0154 [886], and XCU/TC2-2008/0155 [707] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to behave as follows:

a. Report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

b. Disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1133 is applied, clarifying the –**X** option and adding a paragraph to the APPLICATION USAGE section.

Austin Group Defect 1270 is applied, removing the –**n** option from the copy mode SYNOPSIS line.

Austin Group Defect 1278 is applied, removing mention of the –**n** option in connection with write mode.

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

Austin Group Defect 1331 is applied, changing ``st_atime'' to ``st_atim'' and ``st_mtime'' to ``st_mtim''.

Austin Group Defect 1379 is applied, changing the ENVIRONMENT VARIABLES section.

Austin Group Defect 1380 is applied, changing text using the term ``link'' in line with its updated definition and changing the description of the –**u** option.

Austin Group Defect 1618 is applied, adding optional trailing `'s'` and `'S'` characters to the option-argument of the –**s** option.

NAME

pr — print files

SYNOPSIS

XSI
```
pr [+page] [-column] [-adfFmprt] [-e[char][gap]] [-h header]
    [-i[char][gap]] [-l lines] [-n[char][width]] [-o offset] [-s[char]]
    [-w width] [file...]
```

DESCRIPTION

The *pr* utility is a printing and pagination filter. If multiple input files are specified, each shall be read, formatted, and written to standard output. By default, the input shall be separated into 66-line pages, each with:

- A 5-line header that includes the page number, date, time, and the pathname of the file
- A 5-line trailer consisting of blank lines

If standard output is associated with a terminal, diagnostic messages shall be deferred until the *pr* utility has completed processing.

When options specifying multi-column output are specified, output text columns shall be of equal width; input lines that do not fit into a text column shall be truncated. By default, text columns shall be separated with at least one <blank>.

OPTIONS

The *pr* utility shall conform to XBD Section 12.2 (on page 215), except that: the *page* option has a `'+'` delimiter; *page* and *column* can be multi-digit numbers; some of the option-arguments are optional; and some of the option-arguments cannot be specified as separate arguments from the preceding option letter. In particular, the **−s** option does not allow the option letter to be separated from its argument, and the options **−e**, **−i**, and **−n** require that both arguments, if present, not be separated from the option letter.

The following options shall be supported. In the following option descriptions, *column*, *lines*, *offset*, *page*, and *width* are positive decimal integers; *gap* is a non-negative decimal integer.

+*page* Begin output at page number *page* of the formatted input.

−*column* Produce multi-column output that is arranged in *column* columns (the default shall be 1) and is written down each column in the order in which the text is received from the input file. This option should not be used with **−m**. The options **−e** and **−i** shall be assumed for multiple text-column output. Whether or not text columns are produced with identical vertical lengths is unspecified, but a text column shall never exceed the length of the page (see the **−l** option). When used with **−t**, use the minimum number of lines to write the output.

−a Modify the effect of the *−column* option so that the columns are filled across the page in a round-robin order (for example, when *column* is 2, the first input line heads column 1, the second heads column 2, the third is the second line in column 1, and so on).

−d Produce output that is double-spaced; append an extra <newline> following every <newline> found in the input.

−e[*char*][*gap*]
Expand each input <tab> to the next greater column position specified by the formula *n***gap*+1, where *n* is an integer > 0. If *gap* is zero or is omitted, it shall default to 8. All <tab> characters in the input shall be expanded into the appropriate number of <space> characters. If any non-digit character, *char*, is specified, it shall be used as the input <tab>. If the first character of the **−e** option-

111702 argument is a digit, the entire option-argument shall be assumed to be *gap*.

111703 XSI **–f** Use a <form-feed> for new pages, instead of the default behavior that uses a
111704 sequence of <newline> characters. Pause before beginning the first page if the
111705 standard output is associated with a terminal.

111706 **–F** Use a <form-feed> for new pages, instead of the default behavior that uses a
111707 sequence of <newline> characters.

111708 **–h** *header* Use the string *header* to replace the contents of the *file* operand in the page header.

111709 **–i[***char***][***gap***]** In output, replace <space> characters with <tab> characters wherever one or more
111710 adjacent <space> characters reach column positions *gap*+1, 2* *gap*+1, 3* *gap*+1, and
111711 so on. If *gap* is zero or is omitted, default tab settings at every eighth column
111712 position shall be assumed. If any non-digit character, *char*, is specified, it shall be
111713 used as the output <tab>. If the first character of the **–i** option-argument is a digit,
111714 the entire option-argument shall be assumed to be *gap*.

111715 **–l** *lines* Override the 66-line default and reset the page length to *lines*. If *lines* is not greater
111716 than the sum of both the header and trailer depths (in lines), the *pr* utility shall
111717 suppress both the header and trailer, as if the **–t** option were in effect.

111718 **–m** Merge files. Standard output shall be formatted so the *pr* utility writes one line
111719 from each file specified by a *file* operand, side by side into text columns of equal
111720 fixed widths, in terms of the number of column positions. Implementations shall
111721 support merging of at least nine *file* operands.

111722 **–n[***char***][***width***]**
111723 Provide *width*-digit line numbering (default for *width* shall be 5). The number shall
111724 occupy the first *width* column positions of each text column of default output or
111725 each line of **–m** output. If *char* (any non-digit character) is given, it shall be
111726 appended to the line number to separate it from whatever follows (default for *char*
111727 is a <tab>).

111728 **–o** *offset* Each line of output shall be preceded by offset <space> characters. If the **–o** option
111729 is not specified, the default offset shall be zero. The space taken is in addition to the
111730 output line width (see the **–w** option below).

111731 **–p** Pause before beginning each page if the standard output is directed to a terminal;
111732 *pr* shall write an <alert> to standard error and wait for a <newline> to be read on
111733 **/dev/tty**.

111734 **–r** Write no diagnostic reports on failure to open files.

111735 **–s[***char***]** Separate text columns by the single character *char* instead of by the appropriate
111736 number of <space> characters (default for *char* shall be <tab>).

111737 **–t** Write neither the five-line identifying header nor the five-line trailer usually
111738 supplied for each page. Quit writing after the last line of each file without spacing
111739 to the end of the page.

111740 **–w** *width* Set the width of the line to *width* column positions for multiple text-column output
111741 only. If the **–w** option is not specified and the **–s** option is not specified, the default
111742 width shall be 72. If the **–w** option is not specified and the **–s** option is specified,
111743 the default width shall be 512.

111744 For single column output, input lines shall not be truncated.

111745 **OPERANDS**

111746 The following operand shall be supported:

111747 *file* A pathname of a file to be written. If no *file* operands are specified, or if a *file*
111748 operand is `'-'`, the standard input shall be used.

111749 **STDIN**

111750 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is `'-'`.
111751 See the INPUT FILES section.

111752 **INPUT FILES**

111753 The input files shall be text files. If the **−m** option is not specified, an empty input file may, but
111754 should not, be treated as an error.

111755 The file **/dev/tty** shall be used to read responses required by the **−p** option.

111756 **ENVIRONMENT VARIABLES**

111757 The following environment variables shall affect the execution of *pr*:

111758 *LANG* Provide a default value for the internationalization variables that are unset or null.
111759 (See XBD Section 8.2 (on page 169) the precedence of internationalization variables
111760 used to determine the values of locale categories.)

111761 *LC_ALL* If set to a non-empty string value, override the values of all the other
111762 internationalization variables.

111763 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
111764 characters (for example, single-byte as opposed to multi-byte characters in
111765 arguments and input files) and which characters are defined as printable (character
111766 class **print**). Non-printable characters are still written to standard output, but are
111767 not counted for the purpose for column-width and line-length calculations.

111768 *LC_MESSAGES*
111769 Determine the locale that should be used to affect the format and contents of
111770 diagnostic messages written to standard error.

111771 *LC_TIME* Determine the format of the date and time for use in writing header lines.

111772 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

111773 *TZ* Determine the timezone used to calculate date and time strings written in header
111774 lines. If *TZ* is unset or null, an unspecified default timezone shall be used.

111775 **ASYNCHRONOUS EVENTS**

111776 If *pr* receives an interrupt while writing to a terminal, it shall flush all accumulated error
111777 messages to the screen before terminating.

111778 **STDOUT**

111779 If the **−m** option is not specified, the *pr* utility output shall be as follows:

111780 • If an input file is empty and the implementation does not treat this as an error, no output
111781 shall be written for that file and this shall be considered to be successful completion of the
111782 processing for that file.

111783 • For each non-empty input file, the output shall be a paginated version of the original file.

111784 If the **−m** option is specified, the *pr* utility output shall be a paginated version of the merged file
111785 contents, as described in OPTIONS.

111786 In both cases, the pagination shall be accomplished using either <form-feed> characters or a

111787 XSI sequence of <newline> characters, as controlled by the –**F** or –**f** option. Page headers shall be
111788 generated unless the –**t** option is specified, the –**l** option is specified with too small a value (see
111789 OPTIONS), or the –**m** option is specified and all of the input files are empty. The page headers
111790 shall be of the form:

111791 `"\n\n%s %s Page %d\n\n\n", <output of date>, <file>, <page number>`

111792 In the POSIX locale, the <*output of date*> field shall be equivalent to the output of the following
111793 command:

111794 `date "+%b %e %H:%M %Y"`

111795 without the trailing <newline>, as it would appear if executed at the current time if the –**m**
111796 option is specified, or at the following time otherwise:

111797 • The current time on pages being written from standard input.

111798 • The modification time of the file named by the corresponding *file* operand on pages not
111799 being written from standard input.

111800 When the *LC_TIME* locale category is not set to the POSIX locale, a different format and order of
111801 presentation of this field may be used.

111802 If the –**h** option is specified, the <*file*> field shall be replaced by the *header* argument. Otherwise:

111803 • If the –**m** option is specified, the <*file*> field shall be replaced by a null string on all pages.

111804 • If the –**m** option is not specified, the <*file*> field shall be replaced by a null string on pages
111805 containing output that was read from standard input.

111806 **STDERR**

111807 The standard error shall be used for diagnostic messages and for alerting the terminal when –**p**
111808 is specified.

111809 **OUTPUT FILES**

111810 None.

111811 **EXTENDED DESCRIPTION**

111812 None.

111813 **EXIT STATUS**

111814 The following exit values shall be returned:

111815 0 Successful completion.

111816 >0 An error occurred.

111817 **CONSEQUENCES OF ERRORS**

111818 Default.

111819 **APPLICATION USAGE**

111820 A conforming application must protect its first operand, if it starts with a <plus-sign>, by
111821 preceding it with the `"--"` argument that denotes the end of the options. For example, *pr+x*
111822 could be interpreted as an invalid page number or a *file* operand.

111823 If a *file* operand contains <newline>, <form-feed>, or <vertical-tab> characters, or is overly long,
111824 and the *pr* utility is instructed to include the pathname of that file in the header, pagination may
111825 not be handled correctly. Applications can guard against this by using the –**h** option (for
111826 example, passing a sanitized, truncated form of the pathname with –**h**).

EXAMPLES

1. Print a numbered list of all files in the current directory:

   ```
   ls -a | pr -n -h "Files in $(pwd)."
   ```

2. Print **file1** and **file2** as a double-spaced, three-column listing headed by ``file list'':

   ```
   pr -3d -h "file list" file1 file2
   ```

3. Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, . . .:

   ```
   pr -e9 -t <file1 >file2
   ```

RATIONALE

This utility is one of those that does not follow the Utility Syntax Guidelines because of its historical origins. The standard developers could have added new options that obeyed the guidelines (and marked the old options obsolescent) or devised an entirely new utility; there are examples of both actions in this volume of POSIX.1-2024. Because of its widespread use by historical applications, the standard developers decided to exempt this version of *pr* from many of the guidelines.

Implementations are required to accept option-arguments to the **–h**, **–l**, **–o**, and **–w** options whether presented as part of the same argument or as a separate argument to *pr*, as suggested by the Utility Syntax Guidelines. The **–n** and **–s** options, however, are specified as in historical practice because they are frequently specified without their optional arguments. If a <blank> were allowed before the option-argument in these cases, a *file* operand could mistakenly be interpreted as an option-argument in historical applications.

The text about the minimum number of lines in multi-column output was included to ensure that a best effort is made in balancing the length of the columns. There are known historical implementations in which, for example, 60-line files are listed by *pr* –2 as one column of 56 lines and a second of 4. Although this is not a problem when a full page with headers and trailers is produced, it would be relatively useless when used with **–t**.

Historical implementations of the *pr* utility have differed in the action taken for the **–f** option. BSD uses it as described here for the **–F** option; System V uses it to change trailing <newline> characters on each page to a <form-feed> and, if standard output is a TTY device, sends an <alert> to standard error and reads a line from **/dev/tty** before the first page. There were strong arguments from both sides of this issue concerning historical practice and as a result the **–F** option was added. XSI-conformant systems support the System V historical actions for the **–f** option.

The <*output of date*> field in the **–l** format is specified only for the POSIX locale. As noted, the format can be different in other locales. No mechanism for defining this is present in this volume of POSIX.1-2024, as the appropriate vehicle is a message catalog; that is, the format should be specified as a ``message''.

Some implementations of *pr* treat an empty file as an error when **–m** is not specified, but not when **–m** is specified (even if there is only one input file). Implementations are encouraged to eliminate this inconsistency by never treating an empty file as an error.

FUTURE DIRECTIONS

A future version of this standard may require that an empty file is never treated as an error.

SEE ALSO

expand, *lp*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **–p** option is added.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

PASC Interpretation 1003.2-92 #151 (SD5-XCU-ERN-44) is applied.

Austin Group Interpretation 1003.1-2001 #093 is applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 251 is applied, adding a paragraph about problematic pathnames to the APPLICATION USAGE section.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1433 is applied, changing <carriage-return> to <newline> in the description of the **–p** option.

Austin Group Defect 1434 is applied, combining the two option groups in the SYNOPSIS into one.

Austin Group Defect 1590 is applied, clarifying the requirements when an input file is empty and changing the STDOUT section.

NAME

printf — write formatted output

SYNOPSIS

```
printf format [argument...]
```

DESCRIPTION

The *printf* utility shall write formatted operands to the standard output. The *argument* operands shall be formatted under control of the *format* operand.

OPTIONS

None.

OPERANDS

The following operands shall be supported:

format A character string describing the format to use to write the remaining operands. See the EXTENDED DESCRIPTION section.

argument The values to be written to standard output, under the control of *format*. See the EXTENDED DESCRIPTION section.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *printf*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

LC_NUMERIC
Determine the locale for numeric formatting. It shall affect the format of numbers written using the `e`, `E`, `f`, `g`, and `G` conversion specifier characters (if supported).

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

See the EXTENDED DESCRIPTION section.

111932 **STDERR**

111933 The standard error shall be used only for diagnostic messages.

111934 **OUTPUT FILES**

111935 None.

111936 **EXTENDED DESCRIPTION**

111937 The application shall ensure that the *format* operand is a character string, beginning and ending
111938 in its initial shift state, if any. The *format* operand shall be used as the format string described in
111939 XBD Chapter 5 (on page 113) with the following exceptions:

111940 1. A <space> in the format string, in any context other than a flag of a conversion
111941 specification, shall be treated as an ordinary character that is copied to the output.

111942 2. A `'Δ'` character in the format string shall be treated as a `'Δ'` character, not as a <space>.

111943 3. In addition to the escape sequences shown in XBD Chapter 5 (on page 113) (`'\\'`, `'\a'`,
111944 `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`), `"\ddd"`, where *ddd* is a one, two, or three-digit
111945 octal number, shall be written as a byte with the numeric value specified by the octal
111946 number.

111947 4. The implementation shall not precede or follow output from the `d` or `u` conversion
111948 specifiers with <blank> characters not specified by the *format* operand.

111949 5. The implementation shall not precede output from the `o` conversion specifier with zeros
111950 not specified by the *format* operand.

111951 6. The `a`, `A`, `e`, `E`, `f`, `F`, `g`, and `G` conversion specifiers need not be supported.

111952 7. An additional conversion specifier character, `b`, shall be supported as follows. The
111953 argument shall be taken to be a string that can contain <backslash>-escape sequences.
111954 The following <backslash>-escape sequences shall be supported:

111955 — The escape sequences listed in XBD Chapter 5 (on page 113) (`'\\'`, `'\a'`, `'\b'`,
111956 `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`), which shall be converted to the characters they
111957 represent.

111958 — `"\0ddd"`, where *ddd* is a zero, one, two, or three-digit octal number that shall be
111959 converted to a byte with the numeric value specified by the octal number.

111960 — `'\c'`, which shall not be written and shall cause *printf* to ignore any remaining
111961 characters in the string operand containing it, any remaining string operands, and
111962 any additional characters in the *format* operand. If a precision is specified and the
111963 argument contains a `'\c'` after the point at which the number of bytes indicated by
111964 the precision specification have been written, it is unspecified whether the `'\c'`
111965 takes effect.

111966 The interpretation of a <backslash> followed by any other sequence of characters is
111967 unspecified.

111968 Bytes from the converted string shall be written until the end of the string or the number
111969 of bytes indicated by the precision specification is reached. If the precision is omitted, it
111970 shall be taken to be infinite, so all bytes up to the end of the converted string shall be
111971 written.

111972 8. Conversions can be applied to the *n*th *argument* operand rather than to the next *argument*
111973 operand. In this case, the conversion specifier character `'%'` is replaced by the sequence
111974 `"%`*n*`$"`, where *n* is a decimal integer in the range [1,{NL_ARGMAX}], giving the *argument*
111975 operand number. This feature provides for the definition of format strings that select

111976 arguments in an order appropriate to specific languages.

111977 The format can contain either numbered argument conversion specifications (that is, ones
111978 beginning with "`%`*n*`$`"), or unnumbered argument conversion specifications, but not
111979 both. The only exception to this is that "`%%`" can be mixed with the "`%`*n*`$`" form. The
111980 results of mixing numbered and unnumbered argument specifications that consume an
111981 argument are unspecified.

111982 9. For each conversion specification that consumes an argument, an *argument* operand shall
111983 be evaluated and converted to the appropriate type for the conversion as specified below.
111984 The operand to be evaluated shall be determined as follows:

111985 • If the conversion specification begins with a "`%`*n*`$`" sequence, the *n*th *argument*
111986 operand shall be evaluated.

111987 • Otherwise, the evaluated operand shall be the next *argument* operand after the one
111988 evaluated by the previous conversion specification that consumed an argument; if
111989 there is no such previous conversion specification the first *argument* operand shall
111990 be evaluated.

111991 If the *format* operand contains no conversion specifications that consume an argument
111992 and there are *argument* operands present, the results are unspecified.

111993 10. The *format* operand shall be reused as often as necessary to satisfy the *argument* operands.
111994 If conversion specifications beginning with a "`%`*n*`$`" sequence are used, on format reuse
111995 the value of *n* shall refer to the *n*th *argument* operand following the highest numbered
111996 *argument* operand consumed by the previous use of the *format* operand.

111997 11. If an *argument* operand to be consumed by a conversion specification does not exist:

111998 • If it is a numbered argument conversion specification, *printf* should write a
111999 diagnostic message to standard error and exit with non-zero status, but may behave
112000 as for an unnumbered argument conversion specification.

112001 • If it is an unnumbered argument conversion specification, any extra `b`, `c`, or `s`
112002 conversion specifiers shall be evaluated as if a null string argument were supplied
112003 and any other extra conversion specifiers shall be evaluated as if a zero argument
112004 were supplied.

112005 12. If a character sequence in the *format* operand begins with a '`%`' character, but does not
112006 form a valid conversion specification, the behavior is unspecified.

112007 13. The argument to the `c` conversion specifier can be a string containing zero or more bytes.
112008 If it contains one or more bytes, the first byte shall be written and any additional bytes
112009 shall be ignored. If the argument is an empty string, it is unspecified whether nothing is
112010 written or a null byte is written.

112011 The *argument* operands shall be treated as strings if the corresponding conversion specifier is `b`,
112012 `c`, or `s`, and shall be evaluated as if by the *strtod*() function if the corresponding conversion
112013 specifier is `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G`. Otherwise, they shall be evaluated as unsuffixed C integer
112014 constants, as described by the ISO C standard, with the following extensions:

112015 • A leading <plus-sign> or <hyphen-minus> shall be allowed.

112016 • If the leading character is a single-quote or double-quote, the value shall be the numeric
112017 value in the underlying codeset of the character following the single-quote or double-
112018 quote.

- Suffixed integer constants may be allowed.

If an *argument* operand cannot be completely converted into an internal value appropriate to the corresponding conversion specification, a diagnostic message shall be written to standard error and the utility shall not exit with a zero exit status, but shall continue processing any remaining operands and shall write the value accumulated at the time the error was detected to standard output.

It shall not be considered an error if an *argument* operand is not completely used for a `b`, `c`, or `s` conversion.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The floating-point formatting conversion specifications of *printf*( ) are not required because all arithmetic in the shell is integer arithmetic. The *awk* utility performs floating-point calculations and provides its own **printf** function. The *bc* utility can perform arbitrary-precision floating-point arithmetic, but does not provide extensive formatting capabilities. (This *printf* utility cannot really be used to format *bc* output; it does not support arbitrary precision.) Implementations are encouraged to support the floating-point conversions as an extension.

Note that this *printf* utility, like the *printf*( ) function defined in the System Interfaces volume of POSIX.1-2024 on which it is based, makes no special provision for dealing with multi-byte characters when using the `%c` conversion specification or when a precision is specified in a `%b` or `%s` conversion specification. Applications should be extremely cautious using either of these features when there are multi-byte characters in the character set.

No provision is made in this volume of POSIX.1-2024 which allows field widths and precisions to be specified as `'*'` since the `'*'` can be replaced directly in the *format* operand using shell variable substitution. Implementations can also provide this feature as an extension if they so choose.

Hexadecimal character constants as defined in the ISO C standard are not recognized in the *format* operand because there is no consistent way to detect the end of the constant. Octal character constants are limited to, at most, three octal digits, but hexadecimal character constants are only terminated by a non-hex-digit character. In the ISO C standard, string literal concatenation can be used to terminate a constant and follow it with a hexadecimal character to be written. In the shell, similar concatenation can be done using `$'...'` so that the shell converts the hexadecimal sequence before it executes *printf*.

The `%b` conversion specification is not part of the ISO C standard; it has been added here as a portable way to process <backslash>-escapes expanded in string operands as provided by the *echo* utility. See also the APPLICATION USAGE section of *echo* (on page 2811) for ways to use *printf* as a replacement for all of the traditional versions of the *echo* utility.

If an argument cannot be parsed correctly for the corresponding conversion specification, the *printf* utility is required to report an error. Thus, overflow and extraneous characters at the end of an argument being used for a numeric conversion shall be reported as errors.

Unlike the *printf*( ) function, when numbered conversion specifications are used, specifying the

112064 *N*th argument does not require that all the leading arguments, from the first to the (*N–1*)th, are
112065 specified in the format string. For example, `"%3$s %1$d\n"` is an acceptable *format* operand
112066 which evaluates the first and third *argument* operands but not the second.

112067 **EXAMPLES**

112068 To alert the user and then print and read a series of prompts:

```
112069 printf "\aPlease fill in the following: \nName: "
112070 read name
112071 printf "Phone number: "
112072 read phone
```

112073 To read out a list of right and wrong answers from a file, calculate the percentage correctly, and
112074 print them out. The numbers are right-justified and separated by a single <tab>. The percentage
112075 is written to one decimal place of accuracy:

```
112076 while read right wrong ; do
112077     percent=$(echo "scale=1;($right*100)/($right+$wrong)" | bc)
112078     printf "%2d right\t%2d wrong\t(%s%%)\n" \
112079         $right $wrong $percent
112080 done < database_file
```

112081 The command:

```
112082 printf "%5d%4d\n" 1 21 321 4321 54321
```

112083 produces:

```
112084     1  21
112085   3214321
112086 54321   0
```

112087 Note that the *format* operand is used three times to print all of the given strings and that a `'0'`
112088 was supplied by *printf* to satisfy the last `%4d` conversion specification.

112089 The command:

```
112090 printf '%d\n' 10 010 0x10
```

112091 produces:

112092

Output Line	Explanation
112093 `10` 112094	Decimal representation of the numeric value of decimal integer constant 10
112095 `8`	Decimal representation of the numeric value of octal integer constant 010
112096 `16` 112097	Decimal representation of the numeric value of hexadecimal integer constant 0x10

112098 If the implementation supports floating-point conversions, the command:

```
112099 LC_ALL=C printf '%.2f\n' 10 010 0x10 10.1e2 010.1e2 0x10.1p2
```

112100 produces:

Output Line	Explanation
`10.00`	The string `"10"` interpreted as a decimal value, with 2 digits of precision
`10.00`	The string `"010"` interpreted as a decimal (not octal) value, with 2 digits of precision
`16.00`	The string `"0x10"` interpreted as a hexadecimal value, with 2 digits of precision
`1010.00`	The string `"10.1e2"` interpreted as a decimal floating-point value, with 2 digits of precision
`1010.00`	The string `"010.1e2"` interpreted as a decimal (not octal) floating-point value, with 2 digits of precision
`64.25`	The string `"0x10.1p2"` interpreted as a hexadecimal floating-point value, with 2 digits of precision

The *printf* utility is required to notify the user when conversion errors are detected while producing numeric output; thus, the following results would be expected on an implementation with 32-bit two's-complement integers when `%d` is specified as the *format* operand:

Argument	Standard Output	Diagnostic Output
`5a`	`5`	**`printf: "5a" not completely converted`**
`9999999999`	`2147483647`	**`printf: "9999999999" arithmetic overflow`**
`−9999999999`	`−2147483648`	**`printf: "−9999999999" arithmetic overflow`**
`ABC`	`0`	**`printf: "ABC" expected numeric value`**

The diagnostic message format is not specified, but these examples convey the type of information that should be reported. Note that the value shown on standard output is what would be expected as the return value from the *strtol*() function as defined in the System Interfaces volume of POSIX.1-2024. A similar correspondence exists between `%u` and *strtoul*() and `%e`, `%f`, and `%g` (if the implementation supports floating-point conversions) and *strtod*().

In a locale that uses a codeset based on the ISO/IEC 646: 1991 standard, the command:

```
printf "%d\n" 3 +3 −3 \'3 \"+3 "'−3"
```

produces:

Output Line	Explanation
`3`	Decimal representation of the numeric value 3
`3`	Decimal representation of the numeric value +3
`−3`	Decimal representation of the numeric value -3
`51`	Decimal representation of the numeric value of the character `'3'` in the ISO/IEC 646: 1991 standard codeset
`43`	Decimal representation of the numeric value of the character `'+'` in the ISO/IEC 646: 1991 standard codeset
`45`	Decimal representation of the numeric value of the character `'−'` in the ISO/IEC 646: 1991 standard codeset

Since the last two arguments contain more characters than used for the conversion, a diagnostic message is generated and the exit status is non-zero. Note that in a locale with multi-byte characters, the value of a character is intended to be the value of the equivalent of the **wchar_t** representation of the character as described in the System Interfaces volume of POSIX.1-2024.

RATIONALE

The *printf* utility was added to provide functionality that has historically been provided by *echo*. However, due to irreconcilable differences in the various versions of *echo* extant, the version has few special features, leaving those to this new *printf* utility, which is based on one in the Ninth Edition system.

The format strings for the *printf* utility are handled differently than for the *printf*( ) function in several respects. Notable differences include:

- Reuse of the format until all arguments have been consumed.
- No provision for obtaining field width and precision from argument values.
- No requirement to support floating-point conversion specifiers.
- An additional `b` conversion specifier.
- Special handling of leading single-quote or double-quote for integer conversion specifiers.
- Hexadecimal character constants are not recognized in the format.
- Formats that use numbered argument conversion specifications can have gaps in the argument numbers.

Although *printf* implementations have no difficulty handling formats with mixed numbered and unnumbered conversion specifications (unlike the *printf*( ) function where it is undefined behavior), existing implementations differ in behavior. Given the format operand `"%2$d %d\n"`, with some implementations the `"%d"` evaluates the first argument and with others it evaluates the third. Consequently this standard leaves the behavior unspecified (as opposed to undefined).

Earlier versions of this standard specified that arguments for all conversions other than `b`, `c`, and `s` were evaluated in the same way (as C constants, but with stated exceptions). For implementations supporting the floating-point conversions it was not clear whether integer conversions need only accept integer constants and floating-point conversions need only accept floating-point constants, or whether both types of conversions should accept both types of constants. Also by not distinguishing between them, the requirement relating to a leading single-quote or double-quote applied to floating-point conversions even though this provided no useful functionality to applications that was not already available through the integer conversions. The current standard clarifies the situation by specifying that the arguments for floating-point conversions are evaluated as if by *strtod*( ), and the arguments for integer conversions are evaluated as C integer constants, with the special treatment of leading single-

quote and double-quote applying only to integer conversions.

FUTURE DIRECTIONS

A future version of this standard may require that a missing *argument* operand to be consumed by a numbered argument conversion specification is treated as an error.

A future version of this standard is expected to add a `%b` conversion to the *printf*( ) function for binary conversion of integers, in alignment with the next version of the ISO C standard. This will result in an inconsistency between the *printf* utility and *printf*( ) function for format strings containing `%b`. Implementors are encouraged to collaborate on a way to address this which could then be adopted in a future version of this standard. For example, the *printf* utility could add a **–C** option to make the format string behave in the same way, as far as possible, as the *printf*( ) function.

A future version of this standard may add a `%q` conversion to convert a string argument to a quoted output format that can be reused as shell input.

SEE ALSO

awk, *bc*, *echo*

XBD Chapter 5 (on page 113), Chapter 8 (on page 167)

XSH *fprintf*( ), *strtod*( )

CHANGE HISTORY

First released in Issue 4.

Issue 7

Austin Group Interpretations 1003.1-2001 #175 and #177 are applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0156 [727], XCU/TC2-2008/0157 [727,932], XCU/TC2-2008/0158 [584], and XCU/TC2-2008/0159 [727] are applied.

Issue 8

Austin Group Defect 1108 is applied, changing ``twos'' to ``two's''.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1202 is applied, changing the description of how `'\c'` is handled by the `b` conversion specifier.

Austin Group Defects 1209 and 1476 are applied, changing the EXAMPLES section.

Austin Group Defect 1413 is applied, changing the APPLICATION USAGE section.

Austin Group Defect 1562 is applied, clarifying that it is the application's responsibility to ensure that the format is a character string, beginning and ending in its initial shift state, if any.

Austin Group Defect 1592 is applied, adding support for numbered conversion specifications.

Austin Group Defect 1771 is applied, changing the FUTURE DIRECTIONS section.

NAME

prs — print an SCCS file (**DEVELOPMENT**)

SYNOPSIS

XSI
```
prs [-a] [-d dataspec] [-r[SID]] file...
prs [-e|-l] -c cutoff [-d dataspec] file...
prs [-e|-l] -r[SID] [-d dataspec] file...
```

DESCRIPTION

The *prs* utility shall write to standard output parts or all of an SCCS file in a user-supplied format.

OPTIONS

The *prs* utility shall conform to XBD Section 12.2 (on page 215), except that the **–r** option has an optional option-argument. This optional option-argument cannot be presented as a separate argument. The following options shall be supported:

–d *dataspec* Specify the output data specification. The *dataspec* shall be a string consisting of SCCS file *data keywords* (see Data Keywords, on page 3305) interspersed with optional user-supplied text.

–r[*SID***]** Specify the SCCS identification string (SID) of a delta for which information is desired. If no *SID* option-argument is specified, the SID of the most recently created delta shall be assumed.

–e Request information for all deltas created earlier than and including the delta designated via the **–r** option or the date-time given by the **–c** option.

–l Request information for all deltas created later than and including the delta designated via the **–r** option or the date-time given by the **–c** option.

–c *cutoff* Indicate the *cutoff* date-time, in the form:

*YY***[***MM***[***DD***[***HH***[***MM***[***SS***]]]]]**

For the *YY* component, values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.

Note: It is expected that in a future version of this standard the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)

No changes (deltas) to the SCCS file that were created after the specified *cutoff* date-time shall be included in the output. Units omitted from the date-time default to their maximum possible values; for example, **–c 7502** is equivalent to **–c 750228235959**.

–a Request writing of information for both removed—that is, *delta type*=*R* (see *rmdel*)—and existing—that is, *delta type*=*D*,—deltas. If the **–a** option is not specified, information for existing deltas only shall be provided.

OPERANDS

The following operand shall be supported:

file A pathname of an existing SCCS file or a directory. If *file* is a directory, the *prs* utility shall behave as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with **s.**) and unreadable files shall be silently ignored.

If exactly one *file* operand appears, and it is `'−'`, the standard input shall be read; each line of the standard input shall be taken to be the name of an SCCS file to be processed. Non-SCCS files and unreadable files shall be silently ignored.

STDIN

The standard input shall be a text file used only when the *file* operand is specified as `'−'`. Each line of the text file shall be interpreted as an SCCS pathname.

INPUT FILES

Any SCCS files displayed are files of an unspecified format.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *prs*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall be a text file whose format is dependent on the data keywords specified with the **−d** option.

Data Keywords

Data keywords specify which parts of an SCCS file shall be retrieved and output. All parts of an SCCS file have an associated data keyword. A data keyword may appear in a *dataspec* multiple times.

The information written by *prs* shall consist of:

1. The user-supplied text
2. Appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*

The format of a data keyword value shall either be simple (`'S'`), in which keyword substitution is direct, or multi-line (`'M'`).

User-supplied text shall be any text other than recognized data keywords. A <tab> shall be specified by `'\t'` and <newline> by `'\n'`. When the **−r** option is not specified, the default *dataspec* shall be:

```
:PN::\n\n
```

and the following *dataspec* shall be used for each selected delta:


```
:Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:
```

SCCS File Data Keywords

Keyword	Data Item	File Section	Value	Format
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	**:Li:/:Ld:/:Lu:**	S
:Li:	Lines inserted by Delta	"	*nnnnn***	S
:Ld:	Lines deleted by Delta	"	*nnnnn***	S
:Lu:	Lines unchanged by Delta	"	*nnnnn***	S
:DT:	Delta type	"	**D or R**	S
:I:	SCCS ID string (SID)	"	See below**	S
:R:	Release number	"	*nnnn*	S
:L:	Level number	"	*nnnn*	S
:B:	Branch number	"	*nnnn*	S
:S:	Sequence number	"	*nnnn*	S
:D:	Date delta created	"	**:Dy:/:Dm:/:Dd:**	S
:Dy:	Year delta created	"	*nn*	S
:Dm:	Month delta created	"	*nn*	S
:Dd:	Day delta created	"	*nn*	S
:T:	Time delta created	"	**:Th:::Tm:::Ts:**	S
:Th:	Hour delta created	"	*nn*	S
:Tm:	Minutes delta created	"	*nn*	S
:Ts:	Seconds delta created	"	*nn*	S
:P:	Programmer who created Delta	"	*logname*	S
:DS:	Delta sequence number	"	*nnnn*	S
:DP:	Predecessor Delta sequence number	"	*nnnn*	S
:DI:	Sequence number of deltas included, excluded, or ignored	"	**:Dn:/:Dx:/:Dg:**	S
:Dn:	Deltas included (sequence #)	"	**:DS: :DS: . . .**	S
:Dx:	Deltas excluded (sequence #)	"	**:DS: :DS: . . .**	S
:Dg:	Deltas ignored (sequence #)	"	**:DS: :DS: . . .**	S
:MR:	MR numbers for delta	"	*text*	M
:C:	Comments for delta	"	*text*	M
:UN:	User names	User Names	*text*	M
:FL:	Flag list	Flags	*text*	M
:Y:	Module type flag	"	*text*	S
:MF:	MR validation flag	"	**yes** or **no**	S
:MP:	MR validation program name	"	*text*	S
:KF:	Keyword error, warning flag	"	**yes** or **no**	S
:BF:	Branch flag	"	**yes** or **no**	S
:J:	Joint edit flag	"	**yes** or **no**	S
:LK:	Locked releases	"	**:R: . . .**	S
:Q:	User-defined keyword	"	*text*	S
:M:	Module name	"	*text*	S
:FB:	Floor boundary	"	**:R:**	S
:CB:	Ceiling boundary	"	**:R:**	S
:Ds:	Default SID	"	**:I:**	S
:ND:	Null delta flag	"	**yes** or **no**	S
:FD:	File descriptive text	Comments	*text*	M

SCCS File Data Keywords				
Keyword	**Data Item**	**File Section**	**Value**	**Format**
:BD:	Body	Body	*text*	M
:GB:	Gotten body	"	*text*	M
:W:	A form of *what* string	N/A	**:Z::M:\t:I:**	S
:A:	A form of *what* string	N/A	**:Z::Y: :M: :I::Z:**	S
:Z:	*what* string delimiter	N/A	`@(#)`	S
:F:	SCCS filename	N/A	*text*	S
:PN:	SCCS file pathname	N/A	*text*	S

* **:Dt:=:DT: :I: :D: :T: :P: :DS: :DP:**

** **:R:.:L:.:B:.:S:** if the delta is a branch delta (**:BF:==yes**)
:R:.:L: if the delta is not a branch delta (**:BF:==no**)

*** The line statistics are capped at 99 999. For example, if 100 000 lines were unchanged in a certain revision, **:Lu:** shall produce the value 99 999.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

1. The following example:

   ```
   prs -d "User Names for :F: are:\n:UN:" s.file
   ```

   might write to standard output:

   ```
   User Names for s.file are:
   xyz
   131
   abc
   ```

2. The following example:

   ```
   prs -d "Delta for pgm :M:: :I: - :D: By :P:" -r s.file
   ```

   might write to standard output:

   ```
   Delta for pgm main.c: 3.7 - 77/12/01 By cas
   ```


3. As a special case:

   ```
   prs s.file
   ```

   might write to standard output:

   ```
   s.file:
   <blank line>
   D 1.1 77/12/01 00:00:00 cas 1 000000/00000/00000
   MRs:
   bl78-12345
   bl79-54321
   COMMENTS:
   this is the comment line for s.file initial delta
   <blank line>
   ```

   for each delta table entry of the **D** type. The only option allowed to be used with this special case is the –**a** option.

RATIONALE

None.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

admin, *delta*, *get*, *what*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 5

The phrase ``in which keyword substitution is followed by a <newline>'' is deleted from the end of the second paragraph of Data Keywords (on page 3305).

The interpretation of the *YY* component of the –**c** *cutoff* argument is noted.

Issue 6

The normative text is reworded to emphasize the term ``shall'' for implementation requirements.

The Open Group Base Resolution bwg2001-007 is applied, updating the table in STDOUT with a note that line statistics are capped at 99 999 for the **:Li:**, **:Ld:**, **:Lu:**, and **:DL:** keywords.

The Open Group Interpretation PIN4C.00009 is applied.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

112428 Austin Group Defect 1452 is applied, deleting **:KV:** from the list of keywords.

112429 Austin Group Defect 1570 is applied, removing extra spacing in `"=="`.

NAME

ps — report process status

SYNOPSIS

XSI
```
ps [-aAw] [-defl] [-g grouplist] [-G grouplist]
    [-n namelist] [-o format]... [-p proclist] [-t termlist]
    [-u userlist] [-U userlist]
```

DESCRIPTION

The *ps* utility shall write information about processes, subject to having appropriate privileges to obtain information about those processes.

By default, *ps* shall select all processes with the same effective user ID as the current user and the same controlling terminal as the invoker.

OPTIONS

The *ps* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–a Write information for all processes associated with terminals. Implementations may omit session leaders from this list.

–A Write information for all processes.

XSI **–d** Write information for all processes, except session leaders.

XSI **–e** Write information for all processes. (Equivalent to **–A**.)

XSI **–f** Generate a **full** listing. (See the STDOUT section for the contents of a **full** listing.)

XSI **–g** *grouplist* Write information for processes whose session leaders are given in *grouplist*. The application shall ensure that the *grouplist* is a single argument in the form of a <blank> or <comma>-separated list.

–G *grouplist* Write information for processes whose real group ID numbers are given in *grouplist*. The application shall ensure that the *grouplist* is a single argument in the form of a <blank> or <comma>-separated list.

XSI **–l** Generate a **long** listing. (See STDOUT for the contents of a **long** listing.)

XSI **–n** *namelist* Specify the name of an alternative system *namelist* file in place of the default. The name of the default file and the format of a *namelist* file are unspecified.

–o *format* Write information according to the format specification given in *format*. This is fully described in the STDOUT section. Multiple **–o** options can be specified; the format specification shall be interpreted as the <space>-separated concatenation of all the *format* option-arguments.

–p *proclist* Write information for processes whose process ID numbers are given in *proclist*. The application shall ensure that the *proclist* is a single argument in the form of a <blank> or <comma>-separated list.

–t *termlist* Write information for processes associated with terminals given in *termlist*. The application shall ensure that the *termlist* is a single argument in the form of a <blank> or <comma>-separated list. Terminal identifiers shall be given in an XSI implementation-defined format. On XSI-conformant systems, they shall be given in one of two forms: the device's filename (for example, **tty04**) or, if the device's filename starts with **tty**, just the identifier following the characters **tty** (for example, `"04"`).

112473 XSI **–u** *userlist* Write information for processes whose user ID numbers or login names are given
112474 in *userlist*. The application shall ensure that the *userlist* is a single argument in the
112475 form of a <blank> or <comma>-separated list. In the listing, the numerical user ID
112476 shall be written unless the **–f** option is used, in which case the login name shall be
112477 written.

112478 **–U** *userlist* Write information for processes whose real user ID numbers or login names are
112479 given in *userlist*. The application shall ensure that the *userlist* is a single argument
112480 in the form of a <blank> or <comma>-separated list.

112481 **–w** Behave as if the *COLUMNS* environment variable had a value of at least 132. If the
112482 **–w** option is not specified or is specified exactly once, all output lines shall contain
112483 no more than the greater of {LINE_MAX} and *COLUMNS* bytes provided that no
112484 format name is specified multiple times.

112485 XSI With the exception of **–f**, **–l**, **–n** *namelist*, and **–o** *format*, all of the options shown are used to
112486 select processes. If any are specified, the default list shall be ignored and *ps* shall select the
112487 processes represented by the inclusive OR of all the selection-criteria options.

112488 **OPERANDS**

112489 None.

112490 **STDIN**

112491 Not used.

112492 **INPUT FILES**

112493 None.

112494 **ENVIRONMENT VARIABLES**

112495 The following environment variables shall affect the execution of *ps*:

112496 *COLUMNS* Override the system-selected horizontal display line size, used to determine the
112497 number of text columns to display. See XBD Chapter 8 (on page 167) for valid
112498 values and results when it is unset or null.

112499 *LANG* Provide a default value for the internationalization variables that are unset or null.
112500 (See XBD Section 8.2 (on page 169) the precedence of internationalization variables
112501 used to determine the values of locale categories.)

112502 *LC_ALL* If set to a non-empty string value, override the values of all the other
112503 internationalization variables.

112504 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
112505 characters (for example, single-byte as opposed to multi-byte characters in
112506 arguments).

112507 *LC_MESSAGES*
112508 Determine the locale that should be used to affect the format and contents of
112509 diagnostic messages written to standard error and informative messages written to
112510 standard output.

112511 *LC_TIME* Determine the format and contents of the date and time strings displayed.

112512 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

112513 *TZ* Determine the timezone used to calculate date and time strings displayed. If *TZ* is
112514 unset or null, an unspecified default timezone shall be used.

112515 **ASYNCHRONOUS EVENTS**

112516 Default.

112517 **STDOUT**

112518 When the **–o** option is not specified, the standard output format is unspecified.

112519 XSI On XSI-conformant systems, the output format shall be as follows. The column headings and
112520 descriptions of the columns in a *ps* listing are given below. The precise meanings of these fields
112521 are implementation-defined. The letters `'f'` and `'l'` (below) indicate the option (**full** or **long**)
112522 that shall cause the corresponding heading to appear; **all** means that the heading always
112523 appears. Note that these two options determine only what information is provided for a process;
112524 they do not determine which processes are listed.

112525 **F** (l) Flags (octal and additive) associated with the process.
112526 **S** (l) The state of the process.
112527 **UID** (f,l) The user ID number of the process owner; the login name is printed
112528 under the **–f** option.
112529 **PID** (all) The process ID of the process; it is possible to kill a process if this
112530 datum is known.
112531 **PPID** (f,l) The process ID of the parent process.
112532 **C** (f,l) Processor utilization for scheduling.
112533 **PRI** (l) The priority of the process; higher numbers mean lower priority.
112534 **NI** (l) Nice value; used in priority computation.
112535 **ADDR** (l) The address of the process.
112536 **SZ** (l) The size in pages of the total memory requirements of the process,
112537 including text, data, stack, mapped memory and other resources.
112538 **WCHAN** (l) The event for which the process is waiting or sleeping; if blank, the
112539 process is running.
112540 **STIME** (f) Starting time of the process.
112541 **TTY** (all) The controlling terminal for the process.
112542 **TIME** (all) The cumulative execution time for the process.
112543 **CMD** (all) The command name; the full command name and its arguments are
112544 written under the **–f** option.

112545 A process that has exited and has a parent, but has not yet been waited for by the parent, shall be
112546 marked **defunct**.

112547 Under the option **–f**, *ps* tries to determine the command name and arguments given when the
112548 process was created by examining memory or the swap area. Failing this, the command name, as
112549 it would appear without the option **–f**, is written in square brackets.

112550 The **–o** option allows the output format to be specified under user control.

112551 The application shall ensure that the format specification is a list of names presented as a single
112552 argument, <blank> or <comma>-separated. Each variable has a default header. The default
112553 header can be overridden by appending an <equals-sign> and the new text of the header. The
112554 rest of the characters in the argument shall be used as the header text. The fields specified shall
112555 be written in the order specified on the command line, and should be arranged in columns in the
112556 output. The field widths shall be selected by the system to be at least as wide as the header text
112557 (default or overridden value). If the header text is null, such as **–o** *user=*, the field width shall be
112558 at least as wide as the default header text. If all header text fields are null, no header line shall
112559 be written.

112560 The following names are recognized in the POSIX locale:

ruser The real user ID of the process. This shall be the textual user ID, if it can be obtained and the field width permits, or a decimal representation otherwise.

user The effective user ID of the process. This shall be the textual user ID, if it can be obtained and the field width permits, or a decimal representation otherwise.

rgroup The real group ID of the process. This shall be the textual group ID, if it can be obtained and the field width permits, or a decimal representation otherwise.

group The effective group ID of the process. This shall be the textual group ID, if it can be obtained and the field width permits, or a decimal representation otherwise.

pid The decimal value of the process ID.

ppid The decimal value of the parent process ID.

pgid The decimal value of the process group ID.

pcpu The ratio of CPU time used recently to CPU time available in the same period, expressed as a percentage. The meaning of ``recently'' in this context is unspecified. The CPU time available is determined in an unspecified manner.

vsz The size of the process in (virtual) memory in 1 024 byte units as a decimal integer.

nice The decimal value of the nice value of the process; see *nice*.

etime In the POSIX locale, the elapsed time since the process was started, in the form:

[[*dd*−**]***hh*:**]***mm*:*ss*

where *dd* shall represent the number of days, *hh* the number of hours, *mm* the number of minutes, and *ss* the number of seconds. The *dd* field shall be a decimal integer. The *hh*, *mm*, and *ss* fields shall be two-digit decimal integers padded on the left with zeros.

time In the POSIX locale, the cumulative CPU time of the process in the form:

[*dd*−**]***hh*:*mm*:*ss*

The *dd*, *hh*, *mm*, and *ss* fields shall be as described in the **etime** specifier.

tty The name of the controlling terminal of the process (if any) in the same format used by the *who* utility.

comm The name of the command being executed (*argv*[0] value) as a string.

args The command with all its arguments as a string. The implementation may truncate this value to the field width; it is implementation-defined whether any further truncation occurs. It is unspecified whether the string represented is a version of the argument list as it was passed to the command when it started, or is a version of the arguments as they may have been modified by the application. Applications cannot depend on being able to modify their argument list and having that modification be reflected in the output of *ps*.

Any field need not be meaningful in all implementations. In such a case a <hyphen-minus> (**'−'**) should be output in place of the field value.

Only **comm** and **args** shall be allowed to contain <blank> characters; all others shall not. Any implementation-defined variables shall be specified in the system documentation along with the default header and indicating whether the field may contain <blank> characters.

The following table specifies the default header to be used in the POSIX locale corresponding to each format specifier.

112602 **Table 3-19** Variable Names and Default Headers in *ps*

Format Specifier	Default Header	Format Specifier	Default Header
args	**COMMAND**	**ppid**	**PPID**
comm	**COMMAND**	**rgroup**	**RGROUP**
etime	**ELAPSED**	**ruser**	**RUSER**
group	**GROUP**	**time**	**TIME**
nice	**NI**	**tty**	**TT**
pcpu	**%CPU**	**user**	**USER**
pgid	**PGID**	**vsz**	**VSZ**
pid	**PID**		

112612 **STDERR**

112613 The standard error shall be used only for diagnostic messages.

112614 **OUTPUT FILES**

112615 None.

112616 **EXTENDED DESCRIPTION**

112617 None.

112618 **EXIT STATUS**

112619 The following exit values shall be returned:

112620 0 Successful completion.

112621 >0 An error occurred.

112622 **CONSEQUENCES OF ERRORS**

112623 Default.

112624 **APPLICATION USAGE**

112625 Things can change while *ps* is running; the snapshot it gives is only true for an instant, and
112626 might not be accurate by the time it is displayed.

112627 The **args** format specifier is allowed to produce a truncated version of the command arguments.
112628 In some implementations, this information is no longer available when the *ps* utility is executed.

112629 If the field width is too narrow to display a textual ID, the system may use a numeric version.
112630 Normally, the system would be expected to choose large enough field widths, but if a large
112631 number of fields were selected to write, it might squeeze fields to their minimum sizes to fit on
112632 one line. One way to ensure adequate width for the textual IDs is to override the default header
112633 for a field to make it larger than most or all user or group names.

112634 Portable applications should not set *COLUMNS* to a value greater than {LINE_MAX} and should
112635 not specify the **–w** option more than once if the output will be used as input for a utility that
112636 requires a text file as input because lines containing more than {LINE_MAX} bytes may cause
112637 undefined behavior in some implementations of the utility.

112638 There is no special quoting mechanism for header text. The header text is the rest of the
112639 argument. If multiple header changes are needed, multiple **–o** options can be used, such as:

112640
```
ps -o "user=User Name" -o pid=Process\ ID
```

112641 On some implementations, especially multi-level secure systems, *ps* may be severely restricted
112642 and produce information only about child processes owned by the user.

EXAMPLES

The command:

```
ps -o user,pid,ppid=MOM -o args
```

writes at least the following in the POSIX locale:

```
  USER   PID   MOM   COMMAND
helene    34    12   ps -o user,pid,ppid=MOM -o args
```

The contents of the **COMMAND** field need not be the same in all implementations, due to possible truncation.

RATIONALE

There is very little commonality between BSD and System V implementations of *ps*. Many options conflict or have subtly different usages. The standard developers attempted to select a set of options for the base standard that were useful on a wide range of systems and selected options that either can be implemented on both BSD and System V-based systems without breaking the current implementations or where the options are sufficiently similar that any changes would not be unduly problematic for users or implementors.

It is recognized that on some implementations, especially multi-level secure systems, *ps* may be nearly useless. The default output has therefore been chosen such that it does not break historical implementations and also is likely to provide at least some useful information on most systems.

The major change is the addition of the format specification capability. The motivation for this invention is to provide a mechanism for users to access a wider range of system information, if the system permits it, in a portable manner. The fields chosen to appear in this volume of POSIX.1-2024 were arrived at after considering what concepts were likely to be both reasonably useful to the ``average'' user and had a reasonable chance of being implemented on a wide range of systems. Again it is recognized that not all systems are able to provide all the information and, conversely, some may wish to provide more. It is hoped that the approach adopted will be sufficiently flexible and extensible to accommodate most systems. Implementations may be expected to introduce new format specifiers.

The default output should consist of a short listing containing the process ID, terminal name, cumulative execution time, and command name of each process.

The preference of the standard developers would have been to make the format specification an operand of the *ps* command. Unfortunately, BSD usage precluded this.

At one time a format was included to display the environment array of the process. This was deleted because there is no portable way to display it.

The **−A** option is equivalent to the BSD **−g** and the SVID **−e**. Because the two systems differed, a mnemonic compromise was selected.

The **−a** option is described with some optional behavior because the SVID omits session leaders, but BSD does not.

In an early proposal, format specifiers appeared for priority and start time. The former was not defined adequately in this volume of POSIX.1-2024 and was removed in deference to the defined nice value; the latter because elapsed time was considered to be more useful.

In a new BSD version of *ps*, a **−O** option can be used to write all of the default information, followed by additional format specifiers. This was not adopted because the default output is implementation-defined. Nevertheless, this is a useful option that should be reserved for that purpose. In the **−o** option for the POSIX Shell and Utilities *ps*, the format is the concatenation of

each **–o**. Therefore, the user can have an alias or function that defines the beginning of their desired format and add more fields to the end of the output in certain cases where that would be useful.

The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write* require that they all use the same format.

The **pcpu** field indicates that the CPU time available is determined in an unspecified manner. This is because it is difficult to express an algorithm that is useful across all possible machine architectures. Historical counterparts to this value have attempted to show percentage of use in the recent past, such as the preceding minute. Frequently, these values for all processes did not add up to 100%. Implementations are encouraged to provide data in this field to users that will help them identify processes currently affecting the performance of the system.

FUTURE DIRECTIONS

None.

SEE ALSO

kill, *nice*, *renice*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

SD5-XCU-ERN-148 is applied, updating the OPTIONS section.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0160 [584] is applied.

Issue 8

Austin Group Defect 905 is applied, adding the **–w** option.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1141 is applied, changing the description of **SZ** in the STDOUT section.

Austin Group Defect 1175 is applied, changing ``uid'' to ``user'' in the EXAMPLES section.

NAME

pwd — return working directory name

SYNOPSIS

```
pwd [-L|-P]
```

DESCRIPTION

The *pwd* utility shall write to standard output an absolute pathname of the current working directory, which does not contain the filenames dot or dot-dot.

OPTIONS

The *pwd* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported by the implementation:

–L If the *PWD* environment variable contains an absolute pathname of the current directory and the pathname does not contain any components that are dot or dot-dot, *pwd* shall write this pathname to standard output, except that if the *PWD* environment variable is longer than {PATH_MAX} bytes including the terminating null, it is unspecified whether *pwd* writes this pathname to standard output or behaves as if the **–P** option had been specified. Otherwise, the **–L** option shall behave as the **–P** option.

–P The pathname written to standard output shall not contain any components that refer to files of type symbolic link. If there are multiple pathnames that the *pwd* utility could write to standard output, one beginning with a single <slash> character and one or more beginning with two <slash> characters, then it shall write the pathname beginning with a single <slash> character. The pathname shall not contain any unnecessary <slash> characters after the leading one or two <slash> characters.

If both **–L** and **–P** are specified, the last one shall apply. If neither **–L** nor **–P** is specified, the *pwd* utility shall behave as if **–L** had been specified.

OPERANDS

None.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *pwd*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

PWD An absolute pathname of the current working directory. If an application sets or unsets the value of *PWD*, the behavior of *pwd* is unspecified.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The *pwd* utility output is an absolute pathname of the current working directory:

```
"%s\n", <directory pathname>
```

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

If an error is detected other than a write error when writing to standard output, no output shall be written to standard output, a diagnostic message shall be written to standard error, and the exit status shall be non-zero.

APPLICATION USAGE

If the pathname obtained from *pwd* is longer than {PATH_MAX} bytes, it could produce an error if passed to *cd*. Therefore, in order to return to that directory it may be necessary to break the pathname into sections shorter than {PATH_MAX} and call *cd* on each section in turn (the first section being an absolute pathname and subsequent sections being relative pathnames).

EXAMPLES

None.

RATIONALE

Some implementations have historically provided *pwd* as a shell special built-in command.

In most utilities, if an error occurs, partial output may be written to standard output. This does not happen in historical implementations of *pwd* (unless an error condition causes a partial write). Because *pwd* is frequently used in historical shell scripts without checking the exit status, it is important that the historical behavior is required here; therefore, the CONSEQUENCES OF ERRORS section specifically disallows any partial output being written to standard output, except when a write error occurs when writing to standard output.

An earlier version of this standard stated that the *PWD* environment variable was affected when the **–P** option was in effect. This was incorrect; conforming implementations do not do this.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

cd

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *getcwd*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

The **–P** and **–L** options are added to describe actions relating to symbolic links as specified in the IEEE P1003.2b draft standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #097 is applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Changes to the *pwd* utility and *PWD* environment variable have been made to match the changes to the *getcwd*( ) function made for Austin Group Interpretation 1003.1-2001 #140.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0161 [471] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1488 is applied, clarifying the behavior when a write error occurs when writing to standard output.

NAME

read — read from standard input into shell variables

SYNOPSIS

```
read [-r] [-d delim] var...
```

DESCRIPTION

The *read* utility shall read a single logical line from standard input into one or more shell variables.

If the **−r** option is not specified, <backslash> shall act as an escape character. An unescaped <backslash> shall preserve the literal value of a following <backslash> and shall prevent a following byte (if any) from being used to split fields, with the exception of either <newline> or the logical line delimiter specified with the **−d** *delim* option (if it is used and *delim* is not <newline>); it is unspecified which. If this excepted character follows the <backslash>, the *read* utility shall interpret this as line continuation. The <backslash> and the excepted character shall be removed before splitting the input into fields. All other unescaped <backslash> characters shall be removed after splitting the input into fields.

If standard input is a terminal device and the invoking shell is interactive, *read* shall prompt for a continuation line when it reads an input line ending with a <backslash> <newline>, unless the **−r** option is specified.

The terminating logical line delimiter (if any) shall be removed from the input. Then, if the shell variable *IFS* (see Section 2.5.3, on page 2481) is set, and its value is an empty string, the resulting data shall be assigned to the variable named by the first *var* operand, and the variables named by other *var* operands (if any) shall be set to the empty string. No other processing shall be performed in this case.

If *IFS* is unset, or is set to any non-empty value, then a modified version of the field splitting algorithm specified in Section 2.6.5 (on page 2491) shall be applied, with the modifications as follows:

1. The input to the algorithm shall be the logical line (minus terminating delimiter) that was read from standard input, and shall be considered as a single initial field, all of which resulted from expansions, with any escaped byte and the preceding <backslash> escape character treated as if they were the result of a quoted expansion, and all other bytes treated as if they were the results of unquoted expansions.
2. The loop over the contents of that initial field shall cease when either the input is empty or *n* output fields have been generated, where *n* is one less than the number of *var* operands passed to the *read* utility. Any remaining input in the original field being processed shall be returned to the *read* utility ``unsplit''; that is, unmodified except that any leading or trailing *IFS* white space, as defined in Section 2.6.5 (on page 2491), shall be removed.

The specified *var* operands shall be processed in the order they appear on the command line, and the output fields generated by the field splitting algorithm shall be used in the order they were generated, by repeating the following checks until neither is true:

- If more than one *var* operand is yet to be processed and one or more output fields are yet to be used, the variable named by the first unprocessed *var* operand shall be assigned the value of the first unused output field.
- If exactly one *var* operand is yet to be processed and there was some remaining unsplit input returned from the modified field splitting algorithm, the variable named by the unprocessed *var* operand shall be assigned the unsplit input.

If there are still one or more unprocessed *var* operands, each of the variables names by those operands shall be assigned an empty string.

Note that in the case where just one *var* operand is given on the *read* command line, the modified field splitting algorithm ceases after producing zero output fields and simply returns the original input field, with any leading and trailing *IFS* white space removed, as unsplit input. This unsplit input is assigned to the variable named by the *var* operand.

The setting of variables specified by the *var* operands shall affect the current shell execution environment; see Section 2.13 (on page 2522). An error in setting any variable (such as if a *var* has previously been marked *readonly*) shall be considered an error of *read* processing, and shall result in a return value greater than one. Variables named before the one generating the error shall be set as described above; it is unspecified whether variables named later shall be set as above, or *read* simply ceases processing when the error occurs, leaving later named variables unaltered. If *read* is called in a subshell or separate utility execution environment, such as one of the following:

```
(read foo)
nohup read ...
find . -exec read ... \;
```

it shall not affect the shell variables in the caller's environment.

If end-of-file is detected before a terminating logical line delimiter is encountered, the variables specified by the *var* operands shall be set as described above and the exit status shall be 1.

OPTIONS

The *read* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−d *delim* If *delim* consists of one single-byte character, that byte shall be used as the logical line delimiter. If *delim* is the null string, the logical line delimiter shall be the null byte. Otherwise, the behavior is unspecified.

−r Do not treat a <backslash> character in any special way. Consider each <backslash> to be part of the input line.

OPERANDS

The following operand shall be supported:

var The name of an existing or nonexisting shell variable. If a *var* operand names the variable *IFS*, the behavior is unspecified.

If a *var* operand names one of the variables *LANG*, *LC_CTYPE*, or *LC_ALL* and the new value assigned to the variable would change how the bytes in *IFS* form characters, or which characters in *IFS* are considered to be *IFS* white space (see Section 2.6.5, on page 2491), it is unspecified what effects, if any, the change has on how *read* performs field splitting.

STDIN

If the **−d** *delim* option is not specified, or if it is specified and *delim* is not the null string, the standard input shall contain zero or more bytes (which need not form valid characters) and shall not contain any null bytes.

If the **−d** *delim* option is specified and *delim* is the null string, the standard input shall contain zero or more bytes (which need not form valid characters).

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *read*:

IFS Determine the internal field separators used to delimit fields; see Section 2.5.3 (on page 2481).

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

PS2 Provide the prompt string that an interactive shell shall write to standard error when a line ending with a <backslash> <newline> is read and the **–r** option was not specified.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used for diagnostic messages and prompts for continued input.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

1 End-of-file was detected.

>1 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

This utility is required to be intrinsic. See Section 1.7 (on page 2470) for details.

The **−r** option is included to enable *read* to subsume the purpose of the *line* utility, which is not included in POSIX.1-2024.

The **−d** *delim* option enables reading up to an arbitrary single-byte delimiter. When *delim* is the null string, the delimiter is the null byte and this allows *read* to be used to process null-terminated lists of pathnames (as produced by the *find* **−print0** primary), with correct handling of pathnames that contain <newline> characters. Note that in order to specify the null string as the delimiter, **−d** and *delim* need to be specified as two separate arguments. Implementations differ in their handling of <backslash> for line continuation when **−d** *delim* is specified (and *delim* is not <newline>); some treat <backslash>*delim* (or <backslash><NUL> if *delim* is the null string) as a line continuation, whereas others still treat <backslash><newline> as a line continuation. Consequently, portable applications need to specify **−r** whenever they specify **−d** *delim* (and *delim* is not <newline>).

When reading a pathname it is inadvisable to use the contents of the first *var* operand, if non-empty, when the exit status of *read* is 1, as it is likely the result of the command used to generate the list of pathnames (for example *find* with **−print** or **−print0**) being terminated after it has written a partial pathname, and consequently using it could result in the wrong pathname being processed.

Since the *var* operands are processed in the order specified on the command line, if any variable name is specified more than once as a *var* operand, the last assignment made is the one that is in effect when *read* returns, including when an empty string is assigned because no field data was available.

EXAMPLES

The following command:

```
while read -r xx yy
do
    printf "%s %s\n" "$yy" "$xx"
done < input_file
```

prints a file with the first field of each line moved to the end of the line.

RATIONALE

The *read* utility historically has been a shell built-in. It was separated off into its own utility to take advantage of the richer description of functionality introduced by this volume of POSIX.1-2024.

Since *read* affects the current shell execution environment, it is required to be intrinsic. If it is called in a subshell or separate utility execution environment, such as one of the following:

```
(read foo)
nohup read ...
find . -exec read ... \;
```

it does not affect the shell variables in the environment of the caller.

Earlier versions of this standard required the standard input to be a text file, and therefore the results were undefined if the input was not empty and end-of-file was detected before a <newline> character was encountered. However, all of the most popular shell implementations have been found to have consistent behavior in this case, and so the behavior is now specified and the requirement for standard input to be a text file has been relaxed to allow non-empty input that does not end with a <newline>.

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 2 (on page 2472)

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 7

Austin Group Interpretation 1003.1-2001 #194 is applied, clarifying the handling of the <backslash> escape character.

SD5-XCU-ERN-126 is applied, clarifying that input lines end with a <newline>.

The description of here-documents is removed from the *read* reference page.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0162 [958] is applied.

Issue 8

Austin Group Defect 243 is applied, adding the **–d** option and relaxing the requirement for standard input to be a text file.

Austin Group Defect 367 is applied, requiring that *read* distinguishes between detecting end-of-file and an error occurring, setting its exit status to one and greater than one, respectively.

Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that this utility is required to be intrinsic.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1778 is applied, clarifying how field splitting is performed.

Austin Group Defect 1779 is applied, clarifying how an error in setting any variable affects the processing of variables named before or after the one generating the error.

NAME

readlink — display the contents of a symbolic link

SYNOPSIS

```
readlink [-n] file
```

DESCRIPTION

If the *file* operand names a symbolic link, the *readlink* utility shall not follow the symbolic link when resolving *file* and shall write the contents of the symbolic link to standard output. If the **–n** option is not specified, the output to standard output shall be followed by a <newline> character.

If *file* does not name a symbolic link, *readlink* shall write a diagnostic message to standard error and exit with non-zero status.

OPTIONS

The *readlink* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

–n Do not output a trailing <newline> character.

OPERANDS

The following operand shall be supported:

file A pathname of a symbolic link to be read.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *readlink*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

See DESCRIPTION.

113065 **STDERR**

113066 The standard error shall be used only for diagnostic messages.

113067 **OUTPUT FILES**

113068 None.

113069 **EXTENDED DESCRIPTION**

113070 None.

113071 **EXIT STATUS**

113072 The following exit values shall be returned:

113073 0 Successful completion.

113074 >0 An error occurred.

113075 **CONSEQUENCES OF ERRORS**

113076 Default.

113077 **APPLICATION USAGE**

113078 None.

113079 **EXAMPLES**

113080 None.

113081 **RATIONALE**

113082 The *readlink* utility was added because using *ls* **–l** to obtain the contents of a symbolic link is
113083 difficult if the output includes more than one occurrence of the string `" -> "`.

113084 The **–f** option found in many implementations was not included, as the *realpath* utility provides
113085 equivalent functionality with a choice of behaviors.

113086 **FUTURE DIRECTIONS**

113087 None.

113088 **SEE ALSO**

113089 *ln*, *ls*, *realpath*

113090 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

113091 XSH *readlink*( )

113092 **CHANGE HISTORY**

113093 First released in Issue 8.

NAME

realpath — resolve a pathname

SYNOPSIS

```
realpath [-E|-e] file
```

DESCRIPTION

The *realpath* utility shall canonicalize the pathname specified by the *file* operand as follows:

If a call to the *realpath*( ) function with the specified pathname as its first argument would succeed, the canonicalized pathname shall be the pathname that would be returned by that *realpath*( ) call. Otherwise:

- If the **−e** option is specified, the canonicalization shall fail.
- If the **−E** option is specified, then if a call to the *realpath*( ) function with the specified pathname as its first argument would encounter an error condition other than [ENOENT], the canonicalization shall fail; if the call would encounter an [ENOENT] error, *realpath* shall expand all symbolic links that would be encountered in an attempt to resolve the specified pathname using the algorithm specified in XBD Section 4.16 (on page 105), except that any trailing <slash> characters that are not also leading <slash> characters shall be ignored. If this expansion succeeds and the path prefix of the expanded pathname resolves to an existing directory, the canonicalized pathname shall be the expanded pathname. In all other cases, the canonicalization shall fail. If the expanded pathname is not empty, does not begin with a <slash>, and has exactly one pathname component, it shall be treated as if it had a path prefix of `"./"`.
- If no options are specified, *realpath* shall canonicalize the specified pathname in an unspecified manner such that the resulting absolute pathname does not contain any components that refer to files of type symbolic link and does not contain any components that are dot or dot-dot.

Upon successful canonicalization, *realpath* shall write the canonicalized pathname, followed by a <newline> character, to standard output.

If canonicalization fails, or the canonicalized pathname is empty, nothing shall be written to standard output, a diagnostic message shall be written to standard error, and *realpath* shall exit with non-zero status.

OPTIONS

The *realpath* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−E Do not treat it as an error if attempting to resolve the last component of the canonicalized form of the *file* operand results in an [ENOENT] error condition.

−e Treat it as an error if attempting to resolve the last component of the canonicalized form of the *file* operand results in an [ENOENT] error condition.

Specifying more than one of the mutually-exclusive options **−E** and **−e** shall not be considered an error. The last option specified shall determine the behavior of the utility.

OPERANDS

The following operand shall be supported:

file A pathname to be canonicalized.

113136 **STDIN**

113137 Not used.

113138 **INPUT FILES**

113139 None.

113140 **ENVIRONMENT VARIABLES**

113141 The following environment variables shall affect the execution of *realpath*:

113142 *LANG* Provide a default value for the internationalization variables that are unset or null.
113143 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
113144 variables used to determine the values of locale categories.)

113145 *LC_ALL* If set to a non-empty string value, override the values of all the other
113146 internationalization variables.

113147 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
113148 characters (for example, single-byte as opposed to multi-byte characters in
113149 arguments and input files).

113150 *LC_MESSAGES*
113151 Determine the locale that should be used to affect the format and contents of
113152 diagnostic messages written to standard error.

113153 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

113154 **ASYNCHRONOUS EVENTS**

113155 Default.

113156 **STDOUT**

113157 See DESCRIPTION.

113158 **STDERR**

113159 The standard error shall be used only for diagnostic messages.

113160 **OUTPUT FILES**

113161 None.

113162 **EXTENDED DESCRIPTION**

113163 None.

113164 **EXIT STATUS**

113165 The following exit values shall be returned:

113166 0 Successful completion.

113167 >0 An error occurred.

113168 **CONSEQUENCES OF ERRORS**

113169 Default.

APPLICATION USAGE

If neither the **−e** nor the **−E** option is specified, some implementations behave as if **−e** had been specified and others as if **−E** had been specified, but there are also implementations where the behavior differs from both of these. For example, the *mksh* shell has an internal implementation of *realpath* that canonicalizes **/dir/regular_file/..** to **/dir**, whereas the *realpath*( ) function would return an [ENOTDIR] error in this case. Portable applications should always specify either **−e** or **−E**.

EXAMPLES

None.

RATIONALE

The *realpath* utility was added in preference to a **−f** option found in some implementations of the *readlink* utility because it allows the application to specify whether or not a missing final component is to be treated as an error.

The behavior with the **−E** option when *file* does not resolve (with symbolic links followed) to an existing file is not the same as simply calling *realpath*( ) with the path prefix of the *file* operand and writing the resulting pathname, a <slash>, and the last component of *file* to standard output. For example, if **/tmp/nofile** does not exist, and *file* is **A/B** where **A** is an existing directory and **B** is a symbolic link to **/tmp/nofile**, *realpath* with **−E** will output **/tmp/nofile**, but if **B** is a symbolic link to **/tmp/nofile/foo**, *realpath* with **−E** will treat this as an error. In both cases `realpath("A/B")` would fail with *errno* set to [ENOENT]. Even though `realpath("A")` would succeed, in neither case is anything ending **/B** the result.

Trailing <slash> characters (that follow a non-<slash>) are handled differently with **−E** than with **−e**. With **−e** they are handled as for the *realpath*( ) function. With **−E** they are sometimes effectively ignored, and they are never included in the output. For example, if **/tmp/nofile** does not exist and **/tmp/regfile** is an existing regular file:

```
$ realpath -E /tmp/nofile/
/tmp/nofile
$ realpath -E /tmp/regfile/
realpath: /tmp/regfile/: Not a directory
```

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

ln, *ls*, *pwd*, *readlink*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH Section 2.3 (on page 507), *realpath*( )

CHANGE HISTORY

First released in Issue 8.

NAME

renice — set nice values of running processes

SYNOPSIS

```
renice [-g|-p|-u] -n increment ID...
```

DESCRIPTION

The *renice* utility shall request that the nice values (see XBD Section 3.225, on page 64) of one or more running processes be changed. By default, the applicable processes are specified by their process IDs. When a process group is specified (see **−g**), the request shall apply to all processes in the process group.

The nice value shall be bounded in an implementation-defined manner. If the requested *increment* would raise or lower the nice value of the executed utility beyond implementation-defined limits, then the limit whose value was exceeded shall be used.

When a user is *renice*d, the request applies to all processes whose saved set-user-ID matches the user ID corresponding to the user.

Regardless of which options are supplied or any other factor, *renice* shall not alter the nice values of any process unless the user requesting such a change has appropriate privileges to do so for the specified process. If the user lacks appropriate privileges to perform the requested action, the utility shall return an error status.

The saved set-user-ID of the user's process shall be checked instead of its effective user ID when *renice* attempts to determine the user ID of the process in order to determine whether the user has appropriate privileges.

OPTIONS

The *renice* utility shall conform to XBD Section 12.2 (on page 215), except for Guideline 9.

The following options shall be supported:

−g Interpret the following operands as unsigned decimal integer process group IDs.

−n *increment* Specify how the nice value of the specified process or processes is to be adjusted. The *increment* option-argument is a positive or negative decimal integer that shall be used to modify the nice value of the specified process or processes. Negative *increment* values may require appropriate privileges.

−p Interpret the following operands as unsigned decimal integer process IDs. The **−p** option is the default if no options are specified.

−u Interpret the following operands as users. If a user exists with a user name equal to the operand, then the user ID of that user is used in further processing. Otherwise, if the operand represents an unsigned decimal integer, it shall be used as the numeric user ID of the user.

OPERANDS

The following operands shall be supported:

ID A process ID, process group ID, or user name/user ID, depending on the option selected.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *renice*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

1. Adjust the nice value so that process IDs 987 and 32 would have a lower nice value:

```
renice -n 5 -p 987 32
```

2. Adjust the nice value so that group IDs 324 and 76 would have a higher nice value, if the user has appropriate privileges to do so:

```
renice -n -4 -g 324 76
```


3. Adjust the nice value so that numeric user ID 8 and user **sas** would have a lower nice value:

```
renice -n 4 -u 8 sas
```

Useful nice value increments on historical systems include 19 or 20 (the affected processes run only when nothing else in the system attempts to run) and any negative number (to make processes run faster).

RATIONALE

The *gid*, *pid*, and *user* specifications do not fit either the definition of operand or option-argument. However, for clarity, they have been included in the OPTIONS section, rather than the OPERANDS section.

The definition of nice value is not intended to suggest that all processes in a system have priorities that are comparable. Scheduling policy extensions such as the realtime priorities in the System Interfaces volume of POSIX.1-2024 make the notion of a single underlying priority for all scheduling policies problematic. Some implementations may implement the *nice*-related features to affect all processes on the system, others to affect just the general time-sharing activities implied by this volume of POSIX.1-2024, and others may have no effect at all. Because of the use of ``implementation-defined'' in *nice* and *renice*, a wide range of implementation strategies are possible.

Originally, this utility was written in the historical manner, using the term ``nice value''. This was always a point of concern with users because it was never intuitively obvious what this meant. With a newer version of *renice*, which used the term ``system scheduling priority'', it was hoped that novice users could better understand what this utility was meant to do. Also, it would be easier to document what the utility was meant to do. Unfortunately, the addition of the POSIX realtime scheduling capabilities introduced the concepts of process and thread scheduling priorities that were totally unaffected by the *nice*/*renice* utilities or the *nice*()/*setpriority*() functions. Continuing to use the term ``system scheduling priority'' would have incorrectly suggested that these utilities and functions were indeed affecting these realtime priorities. It was decided to revert to the historical term ``nice value'' to reference this unrelated process attribute.

Although this utility has use by system administrators (and in fact appears in the system administration portion of the BSD documentation), the standard developers considered that it was very useful for individual end users to control their own processes.

Earlier versions of this standard allowed the following forms in the SYNOPSIS:

```
renice nice_value[-p] pid...[-g gid...][-p pid...][-u user...]
renice nice_value -g gid...[-g gid...]-p pid...][-u user...]
renice nice_value -u user...[-g gid...]-p pid...][-u user...]
```

These forms are no longer specified by POSIX.1-2024 but may be present in some implementations.

FUTURE DIRECTIONS

None.

SEE ALSO

nice

XBD Section 3.225 (on page 64), Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 5

In the SYNOPSIS, an ellipsis is added to the **–u** option in all three obsolescent forms.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The APPLICATION USAGE section is added.

The obsolescent forms of the SYNOPSIS are removed.

Text previously conditional on POSIX_SAVED_IDS is mandatory in this version. This is a FIPS requirement.

Issue 7

Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The *renice* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1286 is applied, changing the description of the **–n** option.

NAME

rm — remove directory entries

SYNOPSIS

```
rm [-diRrv] file...
rm -f [-diRrv] [file...]
```

DESCRIPTION

The *rm* utility shall remove the directory entry specified by each *file* argument.

If either of the files dot or dot-dot are specified as the basename portion of an operand (that is, the final pathname component) or if an operand resolves to the root directory, *rm* shall write a diagnostic message to standard error and do nothing more with such operands.

For each *file* the following steps shall be taken:

1. If the *file* does not exist:
   a. If the **−f** option is not specified, *rm* shall write a diagnostic message to standard error.
   b. Go on to any remaining *files*.
2. If *file* is of type directory, the following steps shall be taken:
   a. If neither the **−R** option nor the **−r** option is specified, but **−d** is specified, *rm* shall proceed with step 3 for the current file. If none of **−r**, **−R**, or **−d** is specified, *rm* shall write a diagnostic message to standard error, do nothing more with *file*, and go on to any remaining files.
   b. If *file* is an empty directory, *rm* may skip to step 2d. If the **−f** option is not specified, and either the permissions of *file* do not permit writing and the standard input is a terminal or the **−i** option is specified, *rm* shall write a prompt to standard error and read a line from the standard input. If the response is not affirmative, *rm* shall do nothing more with the current file and go on to any remaining files.
   c. For each entry contained in *file*, other than dot or dot-dot, the four steps listed here (1 to 4) shall be taken with the entry as if it were a *file* operand. The *rm* utility shall not traverse directories by following symbolic links into other parts of the hierarchy, but shall remove the links themselves.
   d. If the **−i** option is specified, *rm* shall write a prompt to standard error and read a line from the standard input. If the response is not affirmative, *rm* shall do nothing more with the current file, and go on to any remaining files.
   e. *rm* shall proceed with step 4 for the current file.
3. If the **−f** option is not specified, and either the permissions of *file* do not permit writing and the standard input is a terminal or the **−i** option is specified, *rm* shall write a prompt to the standard error and read a line from the standard input. If the response is not affirmative, *rm* shall do nothing more with the current file and go on to any remaining files.
4. *rm* shall perform actions equivalent to the *remove*() function defined in the System Interfaces volume of POSIX.1-2024 called with a pathname of the current file used as the *path* argument.

   If *rm* successfully performed the above actions on the current file, and the **−v** option is specified, *rm* shall write a message containing the pathname of the current file to the standard output. If the actions fail for any reason, *rm* shall write a diagnostic message to

113397 standard error, do nothing more with the current file, and go on to any remaining files.

113398 The *rm* utility shall be able to descend to arbitrary depths in a file hierarchy, and shall not fail
113399 due to path length limitations (unless an operand specified by the user exceeds system
113400 limitations).

113401 **OPTIONS**

113402 The *rm* utility shall conform to XBD Section 12.2 (on page 215).

113403 The following options shall be supported:

113404 **−d** Remove empty directories. See the DESCRIPTION.

113405 **−f** Do not prompt for confirmation. Do not write diagnostic messages or modify the
113406 exit status in the case of no file operands, or in the case of operands that do not
113407 exist. Any previous occurrences of the **−i** option shall be ignored.

113408 **−i** Prompt for confirmation as described previously. Any previous occurrences of the
113409 **−f** option shall be ignored.

113410 **−R** Remove file hierarchies. See the DESCRIPTION.

113411 **−r** Equivalent to **−R**.

113412 **−v** After each file has been removed, write a message to standard output indicating
113413 that it has been removed.

113414 **OPERANDS**

113415 The following operand shall be supported:

113416 *file* A pathname of a directory entry to be removed.

113417 **STDIN**

113418 The standard input shall be used to read an input line in response to each prompt specified in
113419 the STDOUT section. Otherwise, the standard input shall not be used.

113420 **INPUT FILES**

113421 None.

113422 **ENVIRONMENT VARIABLES**

113423 The following environment variables shall affect the execution of *rm*:

113424 *LANG* Provide a default value for the internationalization variables that are unset or null.
113425 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
113426 variables used to determine the values of locale categories.)

113427 *LC_ALL* If set to a non-empty string value, override the values of all the other
113428 internationalization variables.

113429 *LC_COLLATE*
113430 Determine the locale for the behavior of ranges, equivalence classes, and multi-
113431 character collating elements used in the extended regular expression defined for
113432 the **yesexpr** locale keyword in the *LC_MESSAGES* category.

113433 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
113434 characters (for example, single-byte as opposed to multi-byte characters in
113435 arguments) and the behavior of character classes within regular expressions used
113436 in the extended regular expression defined for the **yesexpr** locale keyword in the
113437 *LC_MESSAGES* category.

LC_MESSAGES
Determine the locale used to process affirmative responses, and the locale used to affect the format and contents of diagnostic messages and prompts written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If the **–v** option is specified, information about each removed file shall be written to standard output in an unspecified format.

STDERR

Prompts shall be written to standard error under the conditions specified in the DESCRIPTION and OPTIONS sections. The prompts shall contain the *file* pathname, but their format is otherwise unspecified. The standard error also shall be used for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All requested directory entries (excluding directory entries where a non-affirmative response was given to a request for confirmation) were successfully deleted. In addition, if the **–v** option is specified, information about each removed directory entry was successfully written to standard output.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *rm* utility is forbidden to remove the names dot and dot-dot in order to avoid the consequences of inadvertently doing something like:

```
rm -r .*
```

Some implementations do not permit the removal of the last hard link to an executable binary file that is being executed; see the [EBUSY] error in the *unlink*( ) function defined in the System Interfaces volume of POSIX.1-2024. Thus, the *rm* utility can fail to remove such files.

The **–i** option causes *rm* to prompt and read the standard input even if the standard input is not a terminal, but in the absence of **–i** the mode prompting is not done when the standard input is not a terminal.

EXAMPLES

1. The following command:

   ```
   rm a.out core
   ```

   removes the directory entries: **a.out** and **core**, or issues an error if either directory entry is itself a directory or does not exist.

2. The following command:

   ```
   rm -Rf junk
   ```

   removes the directory **junk** and all its contents, without prompting.

3. The following command:

   ```
   rm -d name
   ```

   behaves like

   ```
   rmdir name
   ```

   if **name** is a directory (including failing if **name** is not empty), as if by the *rmdir*( ) function; and behaves like

   ```
   rm name
   ```

   if **name** is not a directory, as if by the *unlink*( ) function.

RATIONALE

For absolute clarity, paragraphs (2b) and (3) in the DESCRIPTION of *rm* describing the behavior when prompting for confirmation, should be interpreted in the following manner:

```
if ((NOT f_option) AND
    ((not_writable AND input_is_terminal) OR i_option))
```

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified because implementations may desire more descriptive prompts than those used on historical implementations. Therefore, an application not using the **−f** option, or using the **−i** option, relies on the system to provide the most suitable dialog directly with the user, based on the behavior specified.

The **−r** option is historical practice on all known systems. The synonym **−R** option is provided for consistency with the other utilities in this volume of POSIX.1-2024 that provide options requesting recursive descent through the file hierarchy.

The behavior of the **−f** option in historical versions of *rm* is inconsistent. In general, along with ``forcing'' the unlink without prompting for permission, it always causes diagnostic messages to be suppressed and the exit status to be unmodified for nonexistent operands and files that cannot be unlinked. In some versions, however, the **−f** option suppresses usage messages and system errors as well. Suppressing such messages is not a service to either shell scripts or users.

It is less clear that error messages regarding files that cannot be unlinked (removed) should be suppressed. Although this is historical practice, this volume of POSIX.1-2024 does not permit the **−f** option to suppress such messages.

When given the **−r** and **−i** options, historical versions of *rm* prompt the user twice for each directory, once before removing its contents and once before actually attempting to delete the directory entry that names it. This allows the user to ``prune'' the file hierarchy walk. Historical versions of *rm* were inconsistent in that some did not do the former prompt for directories named on the command line and others had obscure prompting behavior when the **−i** option was specified and the permissions of the file did not permit writing. The POSIX Shell and Utilities *rm* differs little from historic practice, but does require that prompts be consistent. Historical versions of *rm* were also inconsistent in that prompts were done to both standard output and standard error. This volume of POSIX.1-2024 requires that prompts be done to standard error, for consistency with *cp* and *mv*, and to allow historical extensions to *rm* that provide an option to list deleted files on standard output.

The *rm* utility is required to descend to arbitrary depths so that any file hierarchy may be deleted. This means, for example, that the *rm* utility cannot run out of file descriptors during its descent (that is, if the number of file descriptors is limited, *rm* cannot be implemented in the historical fashion where one file descriptor is used per directory level). Also, *rm* is not permitted to fail because of path length restrictions, unless an operand specified by the user is longer than {PATH_MAX}.

The *rm* utility removes symbolic links themselves, not the files they refer to, as a consequence of the dependence on the *unlink*( ) functionality, per the DESCRIPTION. When removing hierarchies with **−r** or **−R**, the prohibition on following symbolic links has to be made explicit.

The addition of the **−d** option allows the use of *rm* to delete either a file or an empty directory without the risk of recursion into a non-empty directory, and without the inherent race between determining a file's type and deciding what action to attempt on that file. If either the **−r** or **−R** option is specified, the use of recursion takes precedence.

The addition of the **−v** option allows a user of *rm* to see which files have been deleted.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

rmdir

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *remove*( ), *rmdir*( ), *unlink*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

Text is added to clarify actions relating to symbolic links as specified in the IEEE P1003.2b draft standard.

Issue 7

Austin Group Interpretations 1003.1-2001 #019 and #091 are applied.

Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the *LC_MESSAGES* environment variable.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0163 [542], XCU/TC2-2008/0164 [819], and XCU/TC2-2008/0165 [542] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 802 is applied, adding the **−d** option.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defects 1154, 1365, and 1487 are applied, adding the **–v** option.

Austin Group Defect 1380 is applied, changing ``last link'' to ``last hard link''.

Austin Group Defect 1732 is applied, changing the EXIT STATUS section.

NAME

rmdel — remove a delta from an SCCS file (**DEVELOPMENT**)

SYNOPSIS

XSI

```
rmdel -r SID file...
```

DESCRIPTION

The *rmdel* utility shall remove the delta specified by the SID from each named SCCS file. The delta to be removed shall be the most recent delta in its branch in the delta chain of each named SCCS file. In addition, the application shall ensure that the SID specified is not that of a version being edited for the purpose of making a delta; that is, if a *p-file* (see *get*) exists for the named SCCS file, the SID specified shall not appear in any entry of the *p-file*.

Removal of a delta shall be restricted to:

1. The user who made the delta
2. The owner of the SCCS file
3. The owner of the directory containing the SCCS file

OPTIONS

The *rmdel* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

−r *SID* Specify the SCCS identification string (*SID*) of the delta to be deleted.

OPERANDS

The following operand shall be supported:

file A pathname of an existing SCCS file or a directory. If *file* is a directory, the *rmdel* utility shall behave as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with **s.**) and unreadable files shall be silently ignored.

If exactly one *file* operand appears, and it is `'−'`, the standard input shall be read; each line of the standard input is taken to be the name of an SCCS file to be processed. Non-SCCS files and unreadable files shall be silently ignored.

STDIN

The standard input shall be a text file used only when the *file* operand is specified as `'−'`. Each line of the text file shall be interpreted as an SCCS pathname.

INPUT FILES

The SCCS files shall be files of unspecified format.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *rmdel*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS
Default.

STDOUT
Not used.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
The SCCS files shall be files of unspecified format. During processing of a *file*, a temporary *x-file*, as described in *admin*, may be created and deleted; a locking *z-file*, as described in *get*, may be created and deleted.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
None.

EXAMPLES
None.

RATIONALE
None.

FUTURE DIRECTIONS
If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO
admin, *delta*, *get*, *prs*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY
First released in Issue 2.

Issue 6
The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

rmdir — remove directories

SYNOPSIS

```
rmdir [-p] dir...
```

DESCRIPTION

The *rmdir* utility shall remove the directory entry specified by each *dir* operand.

For each *dir* operand, the *rmdir* utility shall perform actions equivalent to the *rmdir*( ) function called with the *dir* operand as its only argument.

Directories shall be processed in the order specified. If a directory and a subdirectory of that directory are specified in a single invocation of the *rmdir* utility, the application shall specify the subdirectory before the parent directory so that the parent directory is empty when the *rmdir* utility tries to remove it.

OPTIONS

The *rmdir* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

-p Remove all directories in a pathname. For each *dir* operand:

1. The directory entry it names shall be removed.
2. If the *dir* operand includes more than one pathname component, effects equivalent to the following command shall occur:

   ```
   rmdir -p $(dirname dir)
   ```

OPERANDS

The following operand shall be supported:

dir A pathname of an empty directory to be removed.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *rmdir*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

113694 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

113695 **ASYNCHRONOUS EVENTS**
113696 Default.

113697 **STDOUT**
113698 Not used.

113699 **STDERR**
113700 The standard error shall be used only for diagnostic messages.

113701 **OUTPUT FILES**
113702 None.

113703 **EXTENDED DESCRIPTION**
113704 None.

113705 **EXIT STATUS**
113706 The following exit values shall be returned:

113707 0 Each directory entry specified by a *dir* operand was removed successfully.

113708 >0 An error occurred.

113709 **CONSEQUENCES OF ERRORS**
113710 Default.

113711 **APPLICATION USAGE**
113712 The definition of an empty directory is one that contains, at most, directory entries for dot and
113713 dot-dot.

113714 **EXAMPLES**
113715 If a directory **a** in the current directory is empty except it contains a directory **b** and **a/b** is empty
113716 except it contains a directory **c**:

113717
```
rmdir -p a/b/c
```

113718 removes all three directories.

113719 **RATIONALE**
113720 On historical System V systems, the **–p** option also caused a message to be written to the
113721 standard output. The message indicated whether the whole path was removed or whether part
113722 of the path remained for some reason. The STDERR section requires this diagnostic when the
113723 entire path specified by a *dir* operand is not removed, but does not allow the status message
113724 reporting success to be written as a diagnostic.

113725 The *rmdir* utility on System V also included a **–s** option that suppressed the informational
113726 message output by the **–p** option. This option has been omitted because the informational
113727 message is not specified by this volume of POSIX.1-2024.

113728 **FUTURE DIRECTIONS**
113729 None.

113730 **SEE ALSO**
113731 *rm*

113732 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

113733 XSH *remove*( ), *rmdir*( ), *unlink*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

sact — print current SCCS file-editing activity (**DEVELOPMENT**)

SYNOPSIS

XSI

```
sact file...
```

DESCRIPTION

The *sact* utility shall inform the user of any impending deltas to a named SCCS file by writing a list to standard output. This situation occurs when *get* **−e** has been executed previously without a subsequent execution of *delta*, *unget*, or *sccs* **unedit**.

OPTIONS

None.

OPERANDS

The following operand shall be supported:

file A pathname of an existing SCCS file or a directory. If *file* is a directory, the *sact* utility shall behave as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with **s.**) and unreadable files shall be silently ignored.

If exactly one *file* operand appears, and it is `'−'`, the standard input shall be read; each line of the standard input shall be taken to be the name of an SCCS file to be processed. Non-SCCS files and unreadable files shall be silently ignored.

STDIN

The standard input shall be a text file used only when the *file* operand is specified as `'−'`. Each line of the text file shall be interpreted as an SCCS pathname.

INPUT FILES

Any SCCS files interrogated are files of an unspecified format.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *sact*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The output for each named file shall consist of a line in the following format:

`"%sΔ%sΔ%sΔ%sΔ%s\n", <SID>, <new SID>, <login>, <date>, <time>`

<*SID*> Specifies the SID of a delta that currently exists in the SCCS file to which changes are made to make the new delta.

<*new SID*> Specifies the SID for the new delta to be created.

<*login*> Contains the login name of the user who makes the delta (that is, who executed a *get* for editing).

<*date*> Contains the date that *get* **−e** was executed, in the format used by the *prs* **:D:** data keyword.

<*time*> Contains the time that *get* **−e** was executed, in the format used by the *prs* **:T:** data keyword.

If there is more than one named file or if a directory or standard input is named, each pathname shall be written before each of the preceding lines:

`"\n%s:\n", <pathname>`

STDERR

The standard error shall be used only for optional informative messages concerning SCCS files with no impending deltas, and for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

delta, *get*, *sccs*, *unget*

XBD Chapter 8 (on page 167)

CHANGE HISTORY

First released in Issue 2.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

sccs — front end for the SCCS subsystem (**DEVELOPMENT**)

SYNOPSIS

XSI

```
sccs [-r] [-d path] [-p path] command [options...] [operands...]
```

DESCRIPTION

The *sccs* utility is a front end to the SCCS programs. It also includes the capability to run set-user-id to another user to provide additional protection.

The *sccs* utility shall invoke the specified *command* with the specified *options* and *operands*. By default, each of the *operands* shall be modified by prefixing it with the string `"SCCS/s."`.

The *command* can be the name of one of the SCCS utilities in this volume of POSIX.1-2024 (*admin*, *delta*, *get*, *prs*, *rmdel*, *sact*, *unget*, *val*, or *what*) or one of the pseudo-utilities listed in the EXTENDED DESCRIPTION section.

OPTIONS

The *sccs* utility shall conform to XBD Section 12.2 (on page 215), except that *options* operands are actually options to be passed to the utility named by *command*. When the portion of the command:

```
command [options ... ] [operands ... ]
```

is considered, all of the pseudo-utilities used as *command* shall support the Utility Syntax Guidelines. Any of the other SCCS utilities that can be invoked in this manner support the Guidelines to the extent indicated by their individual OPTIONS sections.

The following options shall be supported preceding the *command* operand:

−d *path* A pathname of a directory to be used as a root directory for the SCCS files. The default shall be the current directory. The **−d** option shall take precedence over the *PROJECTDIR* variable. See **−p**.

−p *path* A pathname of a directory in which the SCCS files are located. The default shall be the **SCCS** directory.

The **−p** option differs from the **−d** option in that the **−d** option-argument shall be prefixed to the entire pathname and the **−p** option-argument shall be inserted before the final component of the pathname. For example:

```
sccs -d /x -p y get a/b
```

converts to:

```
get /x/a/y/s.b
```

This allows the creation of aliases such as:

```
alias syssccs="sccs -d /usr/src"
```

which is used as:

```
syssccs get cmd/who.c
```

−r Invoke *command* with the real user ID of the process, not any effective user ID that the *sccs* utility is set to. Certain commands (*admin*, **check**, **clean**, **diffs**, **info**, *rmdel*, and **tell**) cannot be run set-user-ID by all users, since this would allow anyone to change the authorizations. These commands are always run as the real user.

OPERANDS

The following operands shall be supported:

command An SCCS utility name or the name of one of the pseudo-utilities listed in the EXTENDED DESCRIPTION section.

options An option or option-argument to be passed to *command*.

operands An operand to be passed to *command*.

STDIN

See the utility description for the specified *command*.

INPUT FILES

See the utility description for the specified *command*.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *sccs*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of messages objects and message catalogs.

PROJECTDIR
Provide a default value for the **−d** *path* option. If the value of *PROJECTDIR* begins with a <slash>, it shall be considered an absolute pathname; otherwise, the value of *PROJECTDIR* is treated as a user name and that user's initial working directory shall be examined for a subdirectory **src** or **source**. If such a directory is found, it shall be used. Otherwise, the value shall be used as a relative pathname.

Additional environment variable effects may be found in the utility description for the specified *command*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

See the utility description for the specified *command*.

STDERR

See the utility description for the specified *command*.

OUTPUT FILES

See the utility description for the specified *command*.

EXTENDED DESCRIPTION

The following pseudo-utilities shall be supported as *command* operands. All options referred to in the following list are values given in the *options* operands following *command*.

check Equivalent to **info**, except that nothing shall be printed if nothing is being edited, and a non-zero exit status shall be returned if anything is being edited. The intent is to have this included in an ``install'' entry in a makefile to ensure that everything is included into the SCCS file before a version is installed.

clean Remove everything from the current directory that can be recreated from SCCS files, but do not remove any files being edited. If the **−b** option is given, branches shall be ignored in the determination of whether they are being edited; this is dangerous if branches are kept in the same directory.

create Create an SCCS file, taking the initial contents from the file of the same name. Any options to *admin* are accepted. If the creation is successful, the original files shall be renamed by prefixing the basenames with a comma. These renamed files should be removed after it has been verified that the SCCS files have been created successfully.

delget Perform a *delta* on the named files and then *get* new versions. The new versions shall have ID keywords expanded and shall not be editable. Any **−m**, **−p**, **−r**, **−s**, and **−y** options shall be passed to *delta*, and any **−b**, **−c**, **−e**, **−i**, **−k**, **−l**, **−s**, and **−x** options shall be passed to *get*.

deledit Equivalent to **delget**, except that the *get* phase shall include the **−e** option. This option is useful for making a checkpoint of the current editing phase. The same options shall be passed to *delta* as described above, and all the options listed for *get* above except **−e** shall be passed to **edit**.

diffs Write a difference listing between the current version of the files checked out for editing and the versions in SCCS format. Any **−r**, **−c**, **−i**, **−x**, and **−t** options shall be passed to *get*; any **−l**, **−s**, **−e**, **−f**, **−h**, and **−b** options shall be passed to *diff*. A **−C** option shall be passed to *diff* as **−c**.

edit Equivalent to *get* **−e**.

fix Remove the named delta, but leave a copy of the delta with the changes that were in it. It is useful for fixing small compiler bugs, and so on. The application shall ensure that it is followed by a **−r** *SID* option. Since **fix** does not leave audit trails, it should be used carefully.

info Write a listing of all files being edited. If the **−b** option is given, branches (that is, SIDs with two or fewer components) shall be ignored. If a **−u** *user* option is given, then only files being edited by the named user shall be listed. A **−U** option shall be equivalent to **−u<***current user***>**.

print Write out verbose information about the named files, equivalent to *sccs prs*.

tell Write a **<**newline**>**-separated list of the files being edited to standard output. Takes the **−b**, **−u**, and **−U** options like **info** and **check**.

unedit This is the opposite of an **edit** or a *get* **−e**. It should be used with caution, since any changes made since the *get* are lost.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Many of the SCCS utilities take directory names as operands as well as specific filenames. The pseudo-utilities supported by *sccs* are not described as having this capability, but are not prohibited from doing so.

EXAMPLES

1. To get a file for editing, edit it and produce a new delta:

```
sccs get -e file.c
ex file.c
sccs delta file.c
```

2. To get a file from another directory:

```
sccs -p /usr/src/sccs/s. get cc.c
```

or:

```
sccs get /usr/src/sccs/s.cc.c
```

3. To make a delta of a large number of files in the current directory:

```
sccs delta *.c
```

4. To get a list of files being edited that are not on branches:

```
sccs info -b
```

5. To delta everything being edited by the current user:

```
sccs delta $(sccs tell -U)
```

6. In a makefile, to get source files from an SCCS file if it does not already exist:

```
SRCS = <list of source files>
$(SRCS):
    sccs get $(REL) $@
```

RATIONALE

sccs and its associated utilities are part of the XSI Development Utilities option within the XSI option.

SCCS is an abbreviation for Source Code Control System. It is a maintenance and enhancement tracking tool. When a file is put under SCCS, the source code control system maintains the file and, when changes are made, identifies and stores them in the file with the original source code and/or documentation. As other changes are made, they too are identified and retained in the file.

Retrieval of the original and any set of changes is possible. Any version of the file as it develops can be reconstructed for inspection or additional modification. History data can be stored with each version, documenting why the changes were made, who made them, and when they were made.

FUTURE DIRECTIONS

None.

SEE ALSO

admin, *delta*, *get*, *make*, *prs*, *rmdel*, *sact*, *unget*, *val*, *what*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 6

In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from ``otherwise, the home directory of a user of that name is examined'' to ``otherwise, the value of *PROJECTDIR* is treated as a user name and that user's initial working directory is examined''.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

sed — stream editor

SYNOPSIS

```
sed [-En] script [file...]
sed [-En] -e script [-e script]... [-f script_file]... [file...]
sed [-En] [-e script]... -f script_file [-f script_file]... [file...]
```

DESCRIPTION

The *sed* utility is a stream editor that shall read one or more text files, make editing changes according to a script of editing commands, and write the results to standard output. The script shall be obtained from either the *script* operand string or a combination of the option-arguments from the **−e** *script* and **−f** *script_file* options.

OPTIONS

The *sed* utility shall conform to XBD Section 12.2 (on page 215), except that the order of presentation of the **−e** and **−f** options is significant.

The following options shall be supported:

−E Match using extended regular expressions. Treat each pattern specified as an ERE, as described in XBD Section 9.4 (on page 187).

−e *script* Add the editing commands specified by the *script* option-argument to the end of the script of editing commands.

−f *script_file* Add the editing commands in the file *script_file* to the end of the script of editing commands.

−n Suppress the default output (in which each line, after it is examined for editing, is written to standard output). Only lines explicitly selected for output are written.

If any **−e** or **−f** options are specified, the script of editing commands shall initially be empty. The commands specified by each **−e** or **−f** option shall be added to the script in the order specified. When each addition is made, if the previous addition (if any) was from a **−e** option, a **<**newline**>** shall be inserted before the new addition. The resulting script shall have the same properties as the *script* operand, described in the OPERANDS section.

OPERANDS

The following operands shall be supported:

file A pathname of a file whose contents are read and edited. If multiple *file* operands are specified, the named files shall be read in the order specified and the concatenation shall be edited. If no *file* operands are specified, the standard input shall be used.

script A string to be used as the script of editing commands. The application shall not present a *script* that violates the restrictions of a text file except that the final character need not be a **<**newline**>**.

STDIN

The standard input shall be used if no *file* operands are specified, and shall be used if a *file* operand is `'−'` and the implementation treats the `'−'` as meaning standard input. Otherwise, the standard input shall not be used. See the INPUT FILES section.

INPUT FILES

The input files shall be text files. The *script_file*s named by the **−f** option shall consist of editing commands.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *sed*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), and the behavior of character classes within regular expressions.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The input files shall be written to standard output, with the editing commands specified in the script applied. If the **−n** option is specified, only those input lines selected by the script shall be written to standard output.

STDERR

The standard error shall be used only for diagnostic and warning messages.

OUTPUT FILES

The output files shall be text files whose formats are dependent on the editing commands given.

EXTENDED DESCRIPTION

The *script* shall consist of editing commands of the following form:

[*address***[,***address***]]***function*

where *function* represents a single-character command verb from the list in Editing Commands in sed (on page 3357), followed by any applicable arguments.

The command can be preceded by <blank> characters and/or <semicolon> characters. The function can be preceded by <blank> characters. These optional characters shall have no effect.

In default operation, *sed* cyclically shall append a line of input, less its terminating <newline> character, into the pattern space. Reading from input shall be skipped if a <newline> was in the pattern space prior to a **D** command ending the previous cycle. The *sed* utility shall then apply in sequence all commands whose addresses select that pattern space, until a command starts the next cycle or quits. If no commands explicitly started a new cycle, then at the end of the script the pattern space shall be copied to standard output (except when **−n** is specified) and the

pattern space shall be deleted. Whenever the pattern space is written to standard output or a named file, *sed* shall immediately follow it with a <newline>.

Some of the editing commands use a hold space to save all or part of the pattern space for subsequent retrieval. The pattern and hold spaces shall each be able to hold at least 8 192 bytes.

Addresses in sed

An address is either a decimal number that counts input lines cumulatively across files, a `'$'` character that addresses the last line of input, or a context address. A context address has either the form `"/RE/"` or `"\cREc"`, where RE is a regular expression as described in Regular Expressions in sed, and *c* is any character other than <backslash> or <newline>. In a *sed* context address, the BRE and ERE syntax shall be extended to support escaping occurrences of the <slash> or *c* delimiter within the RE by means of an escape sequence (see XBD Section 9.1, on page 179). For the `"\cREc"` form, if the character designated by *c* is not listed as a special BRE character (if the **−E** option is not specified) or a special ERE character (if **−E** is specified) in XBD Section 9.3.3 (on page 182) or XBD Section 9.4.3 (on page 188), respectively, the escape sequence <backslash>*c* shall be treated as that literal character; otherwise, it is unspecified whether the escape sequence <backslash>*c* is treated as the literal character or the special character. In either case, the escape sequence <backslash>*c* shall not terminate the RE. For example, in the context address `"/abc\/def/"`, the second <slash> stands for itself, so that the RE is `"abc/def"`, and in `"\xabc\xdefx"`, the second `'x'` stands for itself, so that the RE is `"abcxdef"`.

An editing command with no addresses shall select every pattern space.

An editing command with one address shall select each pattern space that matches the address.

An editing command with two addresses shall select the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line shall be selected.) Starting at the first line following the selected range, *sed* shall look again for the first address. Thereafter, the process shall be repeated. Omitting either or both of the address components in the following form produces undefined results:

[*address***[,***address***]]**

Regular Expressions in sed

The *sed* utility shall support the REs described in XBD Chapter 9 (on page 179); by default it shall use BREs as described in XBD Section 9.3 (on page 181), but if the **−E** option is used, it shall use EREs as described in XBD Section 9.4 (on page 187). In *sed*, the BRE and ERE syntax shall be extended as follows:

- The delimiter character that precedes and follows the RE shall not terminate the RE when it appears within a bracket expression, and shall have its normal meaning in the bracket expression. For example, the context address `"\%[%]%"` is equivalent to `"/[%]/"`, and the command `"s-[0-9]--g"` is equivalent to `"s/[0-9]//g"`.
- The escape sequence `'\n'` shall match a <newline> embedded in the pattern space. A literal <newline> shall not be used in the RE of a context address or in the substitute function.
- If an RE is empty (that is, no pattern is specified) *sed* shall behave as if the last RE used in the last command applied (either as an address or as part of a substitute command) was specified.

Editing Commands in sed

In the following list of editing commands, the maximum number of permissible addresses for each function is indicated by [*0addr*], [*1addr*], or [*2addr*], representing zero, one, or two addresses.

The argument *text* shall consist of one or more lines. A <backslash> in the text can be escaped with another <backslash>. The application shall ensure that each embedded <newline> (that is, those other than the terminating <newline> of the last line) in the text is preceded by an unescaped <backslash>. The behavior is unspecified if an unescaped <backslash> is immediately followed by any character other than <backslash> or <newline>, or by the end of a *script*.

The **r** and **w** command verbs, and the *w* flag to the **s** command, take an *rfile* (or *wfile*) parameter, separated from the command verb letter or flag by one or more <blank> characters; implementations may allow zero separation as an extension.

The argument *rfile* or the argument *wfile* shall terminate the editing command. Each *wfile* shall be created before processing begins. Implementations shall support at least ten *wfile* arguments in the script; the actual number (greater than or equal to 10) that is supported by the implementation is unspecified. The use of the *wfile* parameter shall cause that file to be initially created, if it does not exist, or shall replace the contents of an existing file.

The **b**, **r**, **s**, **t**, **w**, **y**, and **:** command verbs shall accept additional arguments. The following synopses indicate which arguments shall be separated from the command verbs by a single <space>.

The **a** and **r** commands schedule text for later output. The text specified for the **a** command, and the contents of the file specified for the **r** command, shall be written to standard output just before the next attempt to fetch a line of input when executing the **c**, **D**, **d**, **N**, or **n** commands, just before executing the **q** command, or when reaching the end of the script. If written when reaching the end of the script, and the –**n** option was not specified, the text shall be written after copying the pattern space to standard output. The contents of the file specified for the **r** command shall be as of the time the output is written, not the time the **r** command is applied. The text shall be output in the order in which the **a** and **r** commands were applied to the input.

Editing commands other than **a**, **b**, **c**, **i**, **r**, **t**, **w**, **:**, and **#** can be followed by a <semicolon>, optional <blank> characters, and another editing command. However, when an **s** editing command is used with the *w* flag, following it with another command in this manner produces undefined results.

A function can be preceded by a `'!'` character, in which case the function shall be applied if the addresses do not select the pattern space. Zero or more <blank> characters shall be accepted before the `'!'` character. It is unspecified whether <blank> characters can follow the `'!'` character, and conforming applications shall not follow the `'!'` character with <blank> characters.

If a *label* argument (to a **b**, **t**, or **:** command) contains characters outside of the portable filename character set, or if a *label* is longer than 8 bytes, the behavior is unspecified. The implementation shall support *label* arguments recognized as unique up to at least 8 bytes; the actual length (greater than or equal to 8) supported by the implementation is unspecified. It is unspecified whether exceeding the maximum supported label length causes an error or a silent truncation.

[*2addr*] {*editing command*
editing command

114181 ...

114182 **}** Execute a list of *sed* editing commands only when the pattern space is selected. The
114183 list of *sed* editing commands shall be surrounded by braces. The braces can be
114184 preceded or followed by <blank> characters. The <right-brace> shall be preceded
114185 by a <newline> or <semicolon> (before any optional <blank> characters preceding
114186 the <right-brace>).

114187 Each command in the list of commands shall be terminated by a <newline>
114188 character, or by a <semicolon> character if permitted when the command is used
114189 outside the braces. The editing commands can be preceded by <blank> characters,
114190 but shall not be followed by <blank> characters.

114191 **[***1addr***]a**
114192 *text* Write text to standard output as described previously.

114193 **[***2addr***]b [***label***]**
114194 Branch to the **:** command verb bearing the *label* argument. If *label* is not specified,
114195 branch to the end of the script.

114196 **[***2addr***]c**
114197 *text* Delete the pattern space. With a 0 or 1 address or at the end of a 2-address range,
114198 place *text* on the output. Start the next cycle.

114199 **[***2addr***]d** Delete the pattern space and start the next cycle.

114200 **[***2addr***]D** If the pattern space contains no <newline>, delete the pattern space and start a
114201 normal new cycle as if the **d** command was issued. Otherwise, delete the initial
114202 segment of the pattern space through the first <newline>, and start the next cycle
114203 with the resultant pattern space and without reading any new input.

114204 **[***2addr***]g** Replace the contents of the pattern space by the contents of the hold space.

114205 **[***2addr***]G** Append to the pattern space a <newline> followed by the contents of the hold
114206 space.

114207 **[***2addr***]h** Replace the contents of the hold space with the contents of the pattern space.

114208 **[***2addr***]H** Append to the hold space a <newline> followed by the contents of the pattern
114209 space.

114210 **[***1addr***]i**
114211 *text* Write *text* to standard output.

114212 **[***2addr***]l** (The letter ell.) Write the pattern space to standard output in a visually
114213 unambiguous form. The characters listed in XBD Table 5-1 (on page 113) (`'\\'`,
114214 `'\a'`, `'\b'`, `'\f'`, `'\r'`, `'\t'`, `'\v'`) shall be written as the corresponding
114215 escape sequence; the `'\n'` in that table is not applicable. Non-printable characters
114216 not in that table shall be written as one three-digit octal number (with a preceding
114217 <backslash>) for each byte in the character (most significant byte first).

114218 Long lines shall be folded, with the point of folding indicated by writing a
114219 <backslash> followed by a <newline>; the length at which folding occurs is
114220 unspecified, but should be appropriate for the output device. The end of each line
114221 shall be marked with a `'$'`.

114222 **[***2addr***]n** Write the pattern space to standard output if the default output has not been
114223 suppressed, and replace the pattern space with the next line of input, less its
114224 terminating <newline>.

If no next line of input is available, the **n** command verb shall branch to the end of the script and quit without starting a new cycle.

[*2addr***]N** Append the next line of input, less its terminating <newline>, to the pattern space, using an embedded <newline> to separate the appended material from the original material. Note that the current line number changes.

If no next line of input is available, the **N** command verb shall branch to the end of the script and quit without starting a new cycle or copying the pattern space to standard output.

[*2addr***]p** Write the pattern space to standard output.

[*2addr***]P** Write the pattern space, up to the first <newline>, to standard output.

[*1addr***]q** Branch to the end of the script and quit without starting a new cycle.

[*1addr***]r** *rfile* Copy the contents of *rfile* to standard output as described previously. If *rfile* does not exist or cannot be read, it shall be treated as if it were an empty file, causing no error condition.

[*2addr***]s/***RE***/***replacement***/***flags***** Substitute the replacement string for instances of the RE in the pattern space. Any character other than <backslash> or <newline> can be used instead of a <slash> to delimit the RE and the replacement. Within the RE (as a *sed* extension to the BRE and ERE syntax) and the replacement, the delimiter shall not terminate the RE or replacement if it is the second character of an escape sequence (see XBD Section 9.1, on page 179). If the delimiter character is not listed as a special BRE character (if the **−E** option is not specified) or a special ERE character (if **−E** is specified) in XBD Section 9.3.3 (on page 182) or XBD Section 9.4.3 (on page 188), respectively, the escaped delimiter shall be treated as that literal character in the RE; otherwise, it is unspecified whether the escaped delimiter is treated as the literal character or the special character. Likewise, if the delimiter character is not <ampersand> (`'&'`), the escaped delimiter shall be treated as that literal character in the replacement; if it is <ampersand>, it is unspecified whether the escaped delimiter is treated as the literal character or the special character (see below).

The replacement string shall be scanned from beginning to end. An <ampersand> (`'&'`) appearing in the replacement shall be replaced by the string matching the RE. The special meaning of `'&'` in this context can be suppressed by preceding it by a <backslash>. The characters `"\`*n*`"`, where *n* is a digit, shall be replaced by the text matched by the corresponding back-reference expression. If the corresponding back-reference expression does not match, then the characters `"\`*n*`"` shall be replaced by the empty string. The special meaning of `"\`*n*`"` where *n* is a digit in this context, can be suppressed by preceding it by a <backslash>. For each other <backslash> encountered, the following character shall lose its special meaning (if any).

A line can be split by substituting a <newline> into it. The application shall escape the <newline> in the replacement by preceding it by a <backslash>.

The meaning of an unescaped <backslash> immediately followed by any character other than `'&'`, <backslash>, a digit, <newline>, or the delimiter character used for this command, is unspecified.

Any <backslash> used to alter the default meaning of a subsequent character shall be discarded from the resulting replacement string. A substitution shall be

114271 considered to have been performed even if the resulting replacement string is
114272 identical to the string that it replaces.

114273 The value of *flags* shall be zero or more of:

114274 *n* Substitute for the *n*th occurrence only of the RE found within the
114275 pattern space.

114276 **g** Globally substitute for all non-overlapping instances of the RE rather
114277 than just the first one. If both **g** and *n* are specified, the results are
114278 unspecified.

114279 **i** Match the regular expression in a case-insensitive way.

114280 **p** Write the pattern space to standard output if a replacement was
114281 made.

114282 **w** *wfile* Write. Append the pattern space to *wfile* if a replacement was made.
114283 A conforming application shall precede the *wfile* argument with one
114284 or more <blank> characters. If the **w** flag is not the last flag value
114285 given in a concatenation of multiple flag values, the results are
114286 undefined.

114287 **[***2addr***]t [***label***]**

114288 Test. Branch to the **:** command verb bearing the *label* if any substitutions have been
114289 made since the most recent reading of an input line or execution of a **t**. If *label* is
114290 not specified, branch to the end of the script.

114291 **[***2addr***]w** *wfile*

114292 Append (write) the pattern space to *wfile*.

114293 **[***2addr***]x** Exchange the contents of the pattern and hold spaces.

114294 **[***2addr***]y/***string1***/***string2***/**

114295 Replace all occurrences of characters in *string1* with the corresponding characters
114296 in *string2*. If a <backslash> followed by an `'n'` appear in *string1* or *string2*, the two
114297 characters shall be handled as a single <newline>. If (after resolving any escape
114298 sequences) the numbers of characters in *string1* and *string2* are not equal, or if any
114299 of the characters in *string1* appear more than once, the results are undefined. Any
114300 character other than <backslash> or <newline> can be used instead of <slash> to
114301 delimit the strings. If the delimiter is not `'n'`, within *string1* and *string2*, the
114302 delimiter itself can be used as a literal character if it is preceded by an unescaped
114303 <backslash>. If a <backslash> character is escaped by an immediately preceding
114304 unescaped <backslash> character in *string1* or *string2*, the two <backslash>
114305 characters shall be treated as a single literal <backslash> character. The meaning of
114306 an unescaped <backslash> followed by any character that is not `'n'`, a
114307 <backslash>, or the delimiter character is undefined.

114308 **[***0addr***]:***label* Do nothing. This command bears a *label* to which the **b** and **t** commands branch.

114309 **[***1addr***]=** Write the following to standard output:

114310 `"%d\n", <`*current line number*`>`

114311 **[***0addr***]** Ignore this empty command.

114312 **[***0addr***]#** Ignore the `'#'` and the remainder of the line (treat them as a comment), with the
114313 single exception that if the first two characters in the script are `"#n"`, the default
114314 output shall be suppressed; this shall be the equivalent of specifying –**n** on the

command line.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Regular expressions match entire strings, not just individual lines, but a <newline> is matched by `'\n'` in a *sed* RE; a <newline> is not allowed by the general definition of regular expression in POSIX.1-2024. Also note that `'\n'` cannot be used to match a <newline> at the end of an arbitrary input line; <newline> characters appear in the pattern space as a result of the **N** editing command.

Applications that use a special RE character as a delimiter (for example, `'.'` or `'*'`) and need to use the delimiter as a literal character in the RE should put it inside a bracket expression, as implementations differ regarding whether escaping it with a <backslash> removes its special meaning. For example, for the context address `"/\.[0-9]/"` to be written with `'.'` as delimiter, the form `"\.[.][0-9]."` needs to be used; `"\.\.[0-9]."` cannot be used portably for this purpose, as it is unspecified whether this would be equivalent to `"/\.[0-9]/"` or `"/.[0-9]/"`. Portable applications cannot use a special RE character as a delimiter if that character needs to have its special meaning in the RE, as escaping it may remove its special meaning.

When using *sed* to process pathnames, it is recommended that LC_ALL, or at least LC_CTYPE and LC_COLLATE, are set to POSIX or C in the environment, since pathnames can contain byte sequences that do not form valid characters in some locales, in which case the utility's behavior would be undefined. In the POSIX locale each byte is a valid single-byte character, and therefore this problem is avoided.

Note that some implementations of *sed* also support an **I** flag for the **s** command as an alias for the lower case **i** flag.

Some implementations of *sed*, when executed in a non-conforming environment, handle <backslash> escapes in regular expressions in a similar way to how *awk* handles them in the lexical token **ERE** (processing `"\t"` as a tab character, etc.). This is a compatible extension except that it conflicts with the requirements of this standard when <backslash> appears inside a bracket expression. A future version of this standard may allow this behavior, and therefore applications should use two <backslash> characters in bracket expressions instead of one in order to ensure future portability. On implementations conforming to the current standard, the second <backslash> is redundant. In the future (and in current non-conforming environments) the first <backslash> may escape the second.

EXAMPLES

This *sed* script simulates the BSD *cat* **–s** command, squeezing excess empty lines from standard input.

```
sed -n '
# Write non-empty lines.
/./ {
    p
    d
```



```
114361          }
114362      # Write a single empty line, then look for more empty lines.
114363      /^$/    p
114364      # Get next line, discard the held <newline> (empty line),
114365      # and look for more empty lines.
114366      :Empty
114367      /^$/    {
114368          N
114369          s/.//
114370          b Empty
114371          }
114372      # Write the non-empty line before going back to search
114373      # for the first in a set of empty lines.
114374          p
114375      '
```

114376 The following *sed* command is a much simpler method of squeezing empty lines, although it is
114377 not quite the same as *cat* **–s** since it removes any initial empty lines:

```
114378      sed –n '/./,/^$/p'
```

114379 **RATIONALE**

114380 This volume of POSIX.1-2024 requires implementations to support at least ten distinct *wfile*s,
114381 matching historical practice on many implementations. Implementations are encouraged to
114382 support more, but conforming applications should not exceed this limit.

114383 The exit status codes specified here are different from those in System V. System V returns 2 for
114384 garbled *sed* commands, but returns zero with its usage message or if the input file could not be
114385 opened. The standard developers considered this to be a bug.

114386 The manner in which the **l** command writes non-printable characters was changed to avoid the
114387 historical backspace-overstrike method, and other requirements to achieve unambiguous output
114388 were added. See the RATIONALE for *ed* for details of the format chosen, which is the same as
114389 that chosen for *sed*.

114390 This volume of POSIX.1-2024 requires implementations to provide pattern and hold spaces of at
114391 least 8 192 bytes, larger than the 4 000 bytes spaces used by some historical implementations, but
114392 less than the 20 480 bytes limit used in an early proposal. Implementations are encouraged to
114393 allocate dynamically larger pattern and hold spaces as needed.

114394 The requirements for acceptance of <blank> and <space> characters in command lines has been
114395 made more explicit than in early proposals to describe clearly the historical practice and to
114396 remove confusion about the phrase ``protect initial blanks [*sic*] and tabs from the stripping that is
114397 done on every script line'' that appears in much of the historical documentation of the *sed* utility
114398 description of text. (Not all implementations are known to have stripped <blank> characters
114399 from text lines, although they all have allowed leading <blank> characters preceding the address
114400 on a command line.)

114401 The treatment of `'#'` comments differs from the SVID which only allows a comment as the first
114402 line of the script, but matches BSD-derived implementations. The comment character is treated
114403 as a command, and it has the same properties in terms of being accepted with leading <blank>
114404 characters; the BSD implementation has historically supported this.

114405 Early proposals required that a *script_file* have at least one non-comment line. Some historical
114406 implementations have behaved in unexpected ways if this were not the case. The standard
114407 developers considered that this was incorrect behavior and that application developers should

not have to avoid this feature. A correct implementation of this volume of POSIX.1-2024 shall permit *script_file*s that consist only of comment lines.

Early proposals indicated that if −**e** and −**f** options were intermixed, all −**e** options were processed before any −**f** options. This has been changed to process them in the order presented because it matches historical practice and is more intuitive.

The characters <backslash> and <newline> cannot be used as RE delimiter characters, as they can never be recognized as the ending delimiter:

- <backslash> does not work, because if it appears unescaped later in the RE, it either escapes the following character, which can then never be the ending delimiter, or it is part of a bracket expression, inside which the ending delimiter for the RE cannot be located.
- <newline> does not work, because if not escaped, it terminates the command, meaning it cannot be the ending delimiter.

Some historical *sed* implementations did not support escaping `'('`, `')'`, `'{'`, and `'}'` when used as a BRE delimiter, as the sequences `"\("` and so on were still treated as special, usually resulting in an error. This standard requires that these sequences are treated as the literal character. This is for consistency with extensions. For example, some implementations treat `"\s"` in a BRE as matching white-space characters, as an extension. This cannot have its special meaning when `'s'` is used as a BRE delimiter in order to ensure portability of *sed* commands that have `'s'` as a delimiter and escape it. If `"\s"` were allowed to keep its special meaning, then the potential for further extensions would mean portable applications would not be able to escape any delimiter character other than <slash>.

The treatment of the **p** flag to the **s** command differs between System V and BSD-based systems when the default output is suppressed. In the two examples:

```
echo a | sed    's/a/A/p'
echo a | sed -n 's/a/A/p'
```

this volume of POSIX.1-2024, BSD, System V documentation, and the SVID indicate that the first example should write two lines with **A**, whereas the second should write one. Some System V systems write the **A** only once in both examples because the **p** flag is ignored if the −**n** option is not specified.

This is a case of a diametrical difference between systems that could not be reconciled through the compromise of declaring the behavior to be unspecified. The SVID/BSD/System V documentation behavior was adopted for this volume of POSIX.1-2024 because:

- No known documentation for any historic system describes the interaction between the **p** flag and the −**n** option.
- The selected behavior is more correct as there is no technical justification for any interaction between the **p** flag and the −**n** option. A relationship between −**n** and the **p** flag might imply that they are only used together, but this ignores valid scripts that interrupt the cyclical nature of the processing through the use of the **D**, **d**, **q**, or branching commands. Such scripts rely on the **p** suffix to write the pattern space because they do not make use of the default output at the ``bottom'' of the script.
- Because the −**n** option makes the **p** flag unnecessary, any interaction would only be useful if *sed* scripts were written to run both with and without the −**n** option. This is believed to be unlikely. It is even more unlikely that programmers have coded the **p** flag expecting it to be unnecessary. Because the interaction was not documented, the likelihood of a programmer discovering the interaction and depending on it is further decreased.

- Finally, scripts that break under the specified behavior produce too much output instead of too little, which is easier to diagnose and correct.

The form of the substitute command that uses the **n** suffix was limited to the first 512 matches in an early proposal. This limit has been removed because there is no reason an editor processing lines of {LINE_MAX} length should have this restriction. The command **s/a/A/2047** should be able to substitute the 2 047th occurrence of **a** on a line.

The **b**, **t**, and **:** commands are documented to ignore leading white space, but no mention is made of trailing white space. Historical implementations of *sed* assigned different locations to the labels `'x'` and `"x "`. This is not useful, and leads to subtle programming errors, but it is historical practice, and changing it could theoretically break working scripts. Implementors are encouraged to provide warning messages about labels that are never referenced by a **b** or **t** command, jumps to labels that do not exist, and label arguments that are subject to truncation.

Earlier versions of this standard allowed for implementations with bytes other than eight bits, but this has been modified in this version.

FUTURE DIRECTIONS

A future version of this standard may allow *sed* to handle <backslash> escapes in regular expressions in a similar way to how *awk* handles them in the lexical token **ERE**. (``Similar'' rather than ``the same'' because *sed* can use BREs or EREs whereas *awk* uses only EREs.)

SEE ALSO

awk, *ed*, *grep*

XBD Table 5-1 (on page 113), Chapter 8 (on page 167), Section 9.3 (on page 181), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- Implementations are required to support at least ten *wfile* arguments in an editing command.

The EXTENDED DESCRIPTION is changed to align with the IEEE P1003.2b draft standard.

IEEE PASC Interpretation 1003.2 #190 is applied.

IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the meaning of the <backslash>-escape sequences in a replacement string for a BRE.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/28 is applied, removing text describing behavior on systems with bytes consisting of more than eight bits.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/29 is applied, making an editorial correction within the Editing Commands in *sed* section.

Issue 7

Austin Group Interpretations 1003.1-2001 #006, #036, and #092 are applied.

SD5-XCU-ERN-97 and SD5-XCU-ERN-123 are applied, updating the SYNOPSIS.

A second example is added.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0133 [262], XCU/TC1-2008/0134 [282,431], XCU/TC1-2008/0135 [269], and XCU/TC1-2008/0136 [282,431] are applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0166 [945], XCU/TC2-2008/0167 [944], XCU/TC2-2008/0168 [945], XCU/TC2-2008/0169 [944], XCU/TC2-2008/0170 [945], XCU/TC2-2008/0171 [533], XCU/TC2-2008/0172 [663], XCU/TC2-2008/0173 [945], and XCU/TC2-2008/0174 [944] are applied.

Issue 8

Austin Group Defect 528 is applied, adding support for selecting the use of EREs instead of BREs, by specifying the –**E** option.

Austin Group Defect 779 is applied, adding the **i** flag to the **s** command.

Austin Group Defect 961 is applied, requiring that **{...}** can be followed by a **<**semicolon**>**, optional **<**blank**>** characters, and another editing command.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1231 is applied, clarifying the handling of **<**backslash**>** in *text* arguments.

Austin Group Defect 1233 is applied, changing the APPLICATION USAGE and FUTURE DIRECTIONS sections.

Austin Group Defect 1319 is applied, changing when the text specified for the **a** command and the contents of the file specified for the **r** command are written.

Austin Group Defect 1550 is applied, clarifying requirements relating to delimiters in context addresses and in **s** and **y** commands.

Austin Group Defect 1578 is applied, clarifying the description of the **y** command.

Austin Group Defect 1767 is applied, clarifying that a **c** command starts the next cycle on every line that its address range matches.

NAME

sh — shell, the standard command language interpreter

SYNOPSIS

OB `sh [-abCefh imnuvx] [-o option]... [+abCefh imnuvx] [+o option]... [command_file [argument...]]`

OB `sh -c [-abCefh imnuvx] [-o option]... [+abCefh imnuvx] [+o option]... command_string [command_name [argument...]]`

OB `sh -s [-abCefh imnuvx] [-o option]... [+abCefh imnuvx] [+o option]... [argument...]`

DESCRIPTION

The *sh* utility is a command language interpreter that shall execute commands read from a command line string, the standard input, or a specified file. The application shall ensure that the commands to be executed are expressed in the language described in Chapter 2 (on page 2472).

Pathname expansion shall not fail due to the size of a file.

Shall input and output redirections have an implementation-defined offset maximum that is established in the open file description.

OPTIONS

The *sh* utility shall conform to XBD Section 12.2 (on page 215), with an extension for support of a leading <plus-sign> (`'+'`) as noted below.

The **−a**, **−b**, **−C**, **−e**, **−f**, **−h**, **−m**, **−n**, **−o** *option*, **−u**, **−v**, and **−x** options are described as part of the *set* utility in Section 2.15 (on page 2526). The option letters derived from the *set* special built-in shall also be accepted with a leading <plus-sign> (`'+'`) instead of a leading <hyphen-minus> (meaning the reverse case of the option as described in this volume of POSIX.1-2024). If the **−o** or **+o** option is specified without an option-argument, the behavior is unspecified.

The following additional options shall be supported:

−c Read commands from the *command_string* operand. Set the value of special parameter 0 (see Section 2.5.2, on page 2479) from the value of the *command_name* operand and the positional parameters ($1, $2, and so on) in sequence from the remaining *argument* operands. No commands shall be read from the standard input.

−i Specify that the shell is *interactive*; see below. An implementation may treat specifying the **−i** option as an error if the real user ID of the calling process does not equal the effective user ID or if the real group ID does not equal the effective group ID.

−s Read commands from the standard input.

If there are no operands and the **−c** option is not specified, the **−s** option shall be assumed.

If the **−i** option is present, or if the shell reads commands from the standard input and the shell's standard input and standard error are attached to a terminal, the shell is considered to be *interactive*.

OPERANDS

The following operands shall be supported:

− A single <hyphen-minus> shall be treated as the first operand and then ignored. If both `'−'` and `"−−"` are given as arguments, or if other operands precede the single <hyphen-minus>, the results are undefined.

argument The positional parameters ($1, $2, and so on) shall be set to *arguments*, if any.

command_file The pathname of a file containing commands. If the pathname contains one or more <slash> characters, the implementation attempts to read that file; the file need not be executable. If the pathname does not contain a <slash> character:

- The implementation shall attempt to read that file from the current working directory; the file need not be executable.
- If the file is not in the current working directory, the implementation may perform a search for an executable file using the value of *PATH*, as described in Section 2.9.1.4 (on page 2502).

Special parameter 0 (see Section 2.5.2, on page 2479) shall be set to the value of *command_file*. If *sh* is called using a synopsis form that omits *command_file*, special parameter 0 shall be set to the value of the first argument passed to *sh* from its parent (for example, *argv*[0] for a C program), which is normally a pathname used to execute the *sh* utility.

command_name
A string assigned to special parameter 0 when executing the commands in *command_string*. If *command_name* is not specified, special parameter 0 shall be set to the value of the first argument passed to *sh* from its parent (for example, *argv*[0] for a C program), which is normally a pathname used to execute the *sh* utility.

command_string
A string that shall be interpreted by the shell as one or more commands, as if the string were the argument to the *system*( ) function defined in the System Interfaces volume of POSIX.1-2024. If the *command_string* operand is an empty string, *sh* shall exit with a zero exit status.

STDIN

The standard input shall be used only if one of the following is true:

- The **–s** option is specified.
- The **–c** option is not specified and no operands are specified.
- The script executes one or more commands that require input from standard input (such as a *read* command that does not redirect its input).

See the INPUT FILES section.

When the shell is using standard input and it invokes a command that also uses standard input, the shell shall ensure that the standard input file pointer points directly after the command it has read when the command begins execution. It shall not read ahead in such a manner that any characters intended to be read by the invoked command are consumed by the shell (whether interpreted by the shell or not) or that characters that are not read by the invoked command are not seen by the shell. When the command expecting to read standard input is started asynchronously by an interactive shell, it is unspecified whether characters are read by the command or interpreted by the shell.

If the standard input to *sh* is a FIFO or terminal device and is set to non-blocking reads, then *sh* shall enable blocking reads on standard input. This shall remain in effect when the command completes.

INPUT FILES

The input file can be of any type, but the initial portion of the file intended to be parsed according to the shell grammar (see Section 2.10.2, on page 2513) shall consist of characters and shall not contain the NUL character. The shell shall not enforce any line length limits. If the input file consists solely of zero or more blank lines and comments, *sh* shall exit with a zero exit status.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *sh*:

UP *ENV* This variable, when and only when an interactive shell is invoked, shall be subjected to parameter expansion (see Section 2.6.2, on page 2485) by the shell, and the resulting value shall be used as a pathname of a file containing shell commands to execute in the current environment. The file need not be executable. If the expanded value of *ENV* is not an absolute pathname, the results are unspecified. *ENV* shall be ignored if the real and effective user IDs or real and effective group IDs of the process are different. The file specified by *ENV* need not be processed if the file can be written by any user other than the user identified by the real (and effective) user ID of the shell process.

UP *FCEDIT* This variable, when expanded by the shell, shall determine the default value for the **−e** *editor* option's *editor* option-argument. If *FCEDIT* is null or unset, *ed* shall be used as the editor.

UP *HISTFILE* Determine a pathname naming a command history file. If the *HISTFILE* variable is not set, the shell may attempt to access or create a file **.sh_history** in the directory referred to by the *HOME* environment variable. If the shell cannot obtain both read and write access to, or create, the history file, it shall use an unspecified mechanism that allows the history to operate properly. (References to history ``file'' in this section shall be understood to mean this unspecified mechanism in such cases.) An implementation may choose to access this variable only when initializing the history file; this initialization shall occur when *fc* or *sh* first attempt to retrieve entries from, or add entries to, the file, as the result of commands issued by the user, the file named by the *ENV* variable, or implementation-defined system start-up files. Implementations may choose to disable the history list mechanism for users with appropriate privileges who do not set *HISTFILE*; the specific circumstances under which this occurs are implementation-defined. If more than one instance of the shell is using the same history file, it is unspecified how updates to the history file from those shells interact. As entries are deleted from the history file, they shall be deleted oldest first. It is unspecified when history file entries are physically removed from the history file.

UP *HISTSIZE* Determine a decimal number representing the limit to the number of previous commands that are accessible. If this variable is unset, an unspecified default greater than or equal to 128 shall be used. The maximum number of commands in the history list is unspecified, but shall be at least 128. An implementation may choose to access this variable only when initializing the history file, as described under *HISTFILE*. Therefore, it is unspecified whether changes made to *HISTSIZE* after the history file has been initialized are effective.

HOME Determine the pathname of the user's home directory. The contents of *HOME* are used in tilde expansion as described in Section 2.6.1 (on page 2485).

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the behavior of range expressions, equivalence classes, and multi-character collating elements within pattern matching.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), which characters are defined as letters (character class **alpha**), and the behavior of character classes within pattern matching.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

UP *MAIL* Determine a pathname of the user's mailbox file for purposes of incoming mail notification. If this variable is set, the shell shall inform the user if the file named by the variable is created or if its modification time has changed. Informing the user shall be accomplished by writing a string of unspecified format to standard error prior to the writing of the next primary prompt string. Such check shall be performed only after the completion of the interval defined by the *MAILCHECK* variable after the last such check. The user shall be informed only if *MAIL* is set and *MAILPATH* is not set.

UP *MAILCHECK*
Establish a decimal integer value that specifies how often (in seconds) the shell shall check for the arrival of mail in the files specified by the *MAILPATH* or *MAIL* variables. The default value shall be 600 seconds. If set to zero, the shell shall check before issuing each primary prompt.

UP *MAILPATH* Provide a list of pathnames and optional messages separated by <colon> characters. If this variable is set, the shell shall inform the user if any of the files named by the variable are created or if any of their modification times change. (See the preceding entry for *MAIL* for descriptions of mail arrival and user informing.) Each pathname can be followed by `'%'` and a string that shall be subjected to parameter expansion and written to standard error when the modification time changes. If a `'%'` character in the pathname is preceded by a <backslash>, it shall be treated as a literal `'%'` in the pathname. The default message is unspecified.

The *MAILPATH* environment variable takes precedence over the *MAIL* variable.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

PATH Establish a string formatted as described in XBD Chapter 8 (on page 167), used to effect command interpretation; see Section 2.9.1.4 (on page 2502).

PWD This variable shall represent an absolute pathname of the current working directory. Assignments to this variable may be ignored.

ASYNCHRONOUS EVENTS

The *sh* utility shall take the standard action for all signals (see Section 1.4, on page 2462) with the following exceptions.

If the shell is interactive, SIGINT signals received during command line editing shall be handled as described in the EXTENDED DESCRIPTION, and SIGINT signals received at other times shall be caught but no action performed.

If the shell is interactive:

- SIGQUIT and SIGTERM signals shall be ignored.
- If the **−m** option is in effect, SIGTTIN, SIGTTOU, and SIGTSTP signals shall be ignored.
- If the **−m** option is not in effect, it is unspecified whether SIGTTIN, SIGTTOU, and SIGTSTP signals are ignored, set to the default action, or caught. If they are caught, the shell shall, in the signal-catching function, set the signal to the default action and raise the signal (after taking any appropriate steps, such as restoring terminal settings).

The standard actions, and the actions described above for interactive shells, can be overridden by use of the *trap* special built-in utility (see *trap* (on page 2565) and Section 2.12, on page 2521).

STDOUT

See the STDERR section.

STDERR

Except as otherwise stated (by the descriptions of any invoked utilities or in interactive mode), standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

UP See Chapter 2. The functionality described in the rest of the EXTENDED DESCRIPTION section shall be provided on implementations that support the User Portability Utilities option (and the rest of this section is not further shaded for this option).

Command History List

When the *sh* utility is being used interactively, it shall maintain a list of commands previously entered from the terminal in the file named by the *HISTFILE* environment variable. The type, size, and internal format of this file are unspecified. Multiple *sh* processes can share access to the file for a user, if file access permissions allow this; see the description of the *HISTFILE* environment variable.

Command Line Editing

When *sh* is being used interactively from a terminal, the current command and the command history (see *fc*) can be edited using *vi*-mode command line editing. This mode uses commands, described below, similar to a subset of those described in the *vi* utility. Implementations may offer other command line editing modes corresponding to other editing utilities.

The command *set* **−o** *vi* shall enable *vi*-mode editing and place *sh* into *vi* insert mode (see Command Line Editing (vi-mode), on page 3371). This command also shall disable any other editing mode that the implementation may provide. The command *set* **+o** *vi* disables *vi*-mode editing.

Certain block-mode terminals may be unable to support shell command line editing. If a terminal is unable to provide either edit mode, it need not be possible to *set* **−o** *vi* when using the

shell on this terminal.

In the following sections, the characters *erase*, *interrupt*, *kill*, and *end-of-file* are those set by the *stty* utility.

Command Line Editing (vi-mode)

In *vi* editing mode, there shall be a distinguished line, the edit line. All the editing operations which modify a line affect the edit line. The edit line is always the newest line in the command history buffer.

With *vi*-mode enabled, *sh* can be switched between insert mode and command mode.

When in insert mode, an entered character shall be inserted into the command line, except as noted in vi Line Editing Insert Mode. Upon entering *sh* and after termination of the previous command, *sh* shall be in insert mode.

Typing an escape character shall switch *sh* into command mode (see vi Line Editing Command Mode, on page 3372). In command mode, an entered character shall either invoke a defined operation, be used as part of a multi-character operation, or be treated as an error. A character that is not recognized as part of an editing command shall terminate any specific editing command and shall alert the terminal. If *sh* receives a SIGINT signal in command mode (whether generated by typing the *interrupt* character or by other means), it shall terminate command line editing on the current command line, reissue the prompt on the next line of the terminal, and reset the command history (see *fc*) so that the most recently executed command is the previous command (that is, the command that was being edited when it was interrupted is not re-entered into the history).

In the following sections, the phrase ``move the cursor to the beginning of the word'' shall mean ``move the cursor to the first character of the current word'' and the phrase ``move the cursor to the end of the word'' shall mean ``move the cursor to the last character of the current word''. The phrase ``beginning of the command line'' indicates the point between the end of the prompt string issued by the shell (or the beginning of the terminal line, if there is no prompt string) and the first character of the command text.

vi Line Editing Insert Mode

While in insert mode, any character typed shall be inserted in the current command line, unless it is from the following set.

<newline> Execute the current command line. If the current command line is not empty, this line shall be entered into the command history (see *fc*).

erase Delete the character previous to the current cursor position and move the current cursor position back one character. In insert mode, characters shall be erased from both the screen and the buffer when backspacing.

interrupt If *sh* receives a SIGINT signal in insert mode (whether generated by typing the *interrupt* character or by other means), it shall terminate command line editing with the same effects as described for interrupting command mode; see Command Line Editing (vi-mode).

kill Clear all the characters from the input line.

<control>-V Insert the next character input, even if the character is otherwise a special insert mode character.

<control>-W Delete the characters from the one preceding the cursor to the preceding word boundary. The word boundary in this case is the closer to the cursor of either the beginning of the line or a character that is in neither the **blank** nor **punct** character classification of the current locale.

end-of-file Interpreted as the end of input in *sh*. This interpretation shall occur only at the beginning of an input line. If *end-of-file* is entered other than at the beginning of the line, the results are unspecified.

<ESC> Place *sh* into command mode.

vi Line Editing Command Mode

In command mode for the command line editing feature, decimal digits not beginning with 0 that precede a command letter shall be remembered. Some commands use these decimal digits as a count number that affects the operation.

The term *motion command* represents one of the commands:

```
<space>  0  b  F  l  W  ^  $  ;  E  f  T  w  |  ,  B  e  h  t
```

If the current line is not the edit line, any command that modifies the current line shall cause the content of the current line to replace the content of the edit line, and the current line shall become the edit line. This replacement cannot be undone (see the **u** and **U** commands below). The modification requested shall then be performed to the edit line. When the current line is the edit line, the modification shall be done directly to the edit line.

Any command that is preceded by *count* shall take a count (the numeric value of any preceding decimal digits). Unless otherwise noted, this count shall cause the specified operation to repeat by the number of times specified by the count. Also unless otherwise noted, a *count* that is out of range is considered an error condition and shall alert the terminal, but neither the cursor position, nor the command line, shall change.

The terms *word* and *bigword* are used as defined in the *vi* description. The term *save buffer* corresponds to the term *unnamed buffer* in *vi*.

The following commands shall be recognized in command mode:

<newline> Execute the current command line. If the current command line is not empty, this line shall be entered into the command history (see *fc*).

<control>-L Redraw the current command line. Position the cursor at the same location on the redrawn line.

Insert the character `'#'` at the beginning of the current command line and treat the resulting edit line as a comment. This line shall be entered into the command history; see *fc*.

= Display the possible shell word expansions (see Section 2.6, on page 2483) of the bigword at the current command line position.

Note: This does not modify the content of the current line, and therefore does not cause the current line to become the edit line.

These expansions shall be displayed on subsequent terminal lines. If the bigword contains none of the characters `'?'`, `'*'`, or `'['`, an <asterisk> (`'*'`) shall be implicitly assumed at the end. If any directories are matched, these expansions shall have a `'/'` character appended. After the expansion, the line shall be redrawn, the cursor repositioned at the current cursor position, and *sh* shall be placed in command mode.

\ Perform pathname expansion (see Section 2.6.6, on page 2493) on the current bigword, up to the largest set of characters that can be matched uniquely. If the bigword contains none of the characters `'?'`, `'*'`, or `'['`, an <asterisk> (`'*'`) shall be implicitly assumed at the end. This maximal expansion then shall replace the original bigword in the command line, and the cursor shall be placed after this expansion. If the resulting bigword completely and uniquely matches a directory, a `'/'` character shall be inserted directly after the bigword. If some other file is completely matched, a single <space> shall be inserted after the bigword. After this operation, *sh* shall be placed in insert mode.

* Perform pathname expansion on the current bigword and insert all expansions into the command to replace the current bigword, with each expansion separated by a single <space>. If at the end of the line, the current cursor position shall be moved to the first column position following the expansions and *sh* shall be placed in insert mode. Otherwise, the current cursor position shall be the last column position of the first character after the expansions and *sh* shall be placed in insert mode. If the current bigword contains none of the characters `'?'`, `'*'`, or `'['`, before the operation, an <asterisk> (`'*'`) shall be implicitly assumed at the end.

@*letter* Insert the value of the alias named *_letter*. The symbol *letter* represents a single alphabetic character from the portable character set; implementations may support additional characters as an extension. If the alias *_letter* contains other editing commands, these commands shall be performed as part of the insertion. If no alias *_letter* is enabled, this command shall have no effect.

[*count*]**~** Convert, if the current character is a lowercase letter, to the equivalent uppercase letter and *vice versa*, as prescribed by the current locale. The current cursor position then shall be advanced by one character. If the cursor was positioned on the last character of the line, the case conversion shall occur, but the cursor shall not advance. If the `'~'` command is preceded by a *count*, that number of characters shall be converted, and the cursor shall be advanced to the character position after the last character converted. If the *count* is larger than the number of characters after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count*]**.** Repeat the most recent non-motion command, even if it was executed on an earlier command line. If the previous command was preceded by a *count*, and no count is given on the `'.'` command, the count from the previous command shall be included as part of the repeated command. If the `'.'` command is preceded by a *count*, this shall override any *count* argument to the previous command. The *count* specified in the `'.'` command shall become the count for subsequent `'.'` commands issued without a count.

[*number*]**v** Invoke the *vi* editor to edit the current command line in a temporary file. When the editor exits, the commands in the temporary file shall be executed and placed in the command history. If a *number* is included, it specifies the command number in the command history to be edited, rather than the current command line.

[*count*]**l** (ell)
[*count*]<space>
Move the current cursor position to the next character position. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the *count* is larger than the number of characters after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count***]h** Move the current cursor position to the *count*th (default 1) previous character position. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the count is larger than the number of characters before the cursor, this shall not be considered an error; the cursor shall move to the first character on the line.

[*count***]w** Move to the start of the next word. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the *count* is larger than the number of words after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count***]W** Move to the start of the next bigword. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the *count* is larger than the number of bigwords after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count***]e** Move to the end of the current word. If at the end of a word, move to the end of the next word. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the *count* is larger than the number of words after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count***]E** Move to the end of the current bigword. If at the end of a bigword, move to the end of the next bigword. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the *count* is larger than the number of bigwords after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count***]b** Move to the beginning of the current word. If at the beginning of a word, move to the beginning of the previous word. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the *count* is larger than the number of words preceding the cursor, this shall not be considered an error; the cursor shall return to the first character on the line.

[*count***]B** Move to the beginning of the current bigword. If at the beginning of a bigword, move to the beginning of the previous bigword. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the *count* is larger than the number of bigwords preceding the cursor, this shall not be considered an error; the cursor shall return to the first character on the line.

^ Move the current cursor position to the first character on the input line that is not a <blank>.

$ Move to the last character position on the current command line.

0 (Zero.) Move to the first character position on the current command line.

[*count***] |** Move to the *count*th character position on the current command line. If no number is specified, move to the first position. The first character position shall be numbered 1. If the count is larger than the number of characters on the line, this shall not be considered an error; the cursor shall be placed on the last character on the line.

114917 [*count*]**f***c* Move to the first occurrence of the character `'c'` that occurs after the current
114918 cursor position. If the cursor was positioned on the last character of the line, the
114919 terminal shall be alerted and the cursor shall not be advanced. If the character `'c'`
114920 does not occur in the line after the current cursor position, the terminal shall be
114921 alerted and the cursor shall not be moved.

114922 [*count*]**F***c* Move to the first occurrence of the character `'c'` that occurs before the current
114923 cursor position. If the cursor was positioned on the first character of the line, the
114924 terminal shall be alerted and the cursor shall not be moved. If the character `'c'`
114925 does not occur in the line before the current cursor position, the terminal shall be
114926 alerted and the cursor shall not be moved.

114927 [*count*]**t***c* Move to the character before the first occurrence of the character `'c'` that occurs
114928 after the current cursor position. If the cursor was positioned on the last character
114929 of the line, the terminal shall be alerted and the cursor shall not be advanced. If the
114930 character `'c'` does not occur in the line after the current cursor position, the
114931 terminal shall be alerted and the cursor shall not be moved.

114932 [*count*]**T***c* Move to the character after the first occurrence of the character `'c'` that occurs
114933 before the current cursor position. If the cursor was positioned on the first
114934 character of the line, the terminal shall be alerted and the cursor shall not be
114935 moved. If the character `'c'` does not occur in the line before the current cursor
114936 position, the terminal shall be alerted and the cursor shall not be moved.

114937 [*count*]**;** Repeat the most recent **f**, **F**, **t**, or **T** command. Any number argument on that
114938 previous command shall be ignored. Errors are those described for the repeated
114939 command.

114940 [*count*]**,** Repeat the most recent **f**, **F**, **t**, or **T** command. Any number argument on that
114941 previous command shall be ignored. However, reverse the direction of that
114942 command.

114943 **a** Enter insert mode after the current cursor position. Characters that are entered
114944 shall be inserted before the next character.

114945 **A** Enter insert mode after the end of the current command line.

114946 **i** Enter insert mode at the current cursor position. Characters that are entered shall
114947 be inserted before the current character.

114948 **I** Enter insert mode at the beginning of the current command line.

114949 **R** Enter insert mode, replacing characters from the command line beginning at the
114950 current cursor position.

114951 [*count*]**c***motion*
114952 Delete the characters between the current cursor position and the cursor position
114953 that would result from the specified motion command. Then enter insert mode
114954 before the first character following any deleted characters. If *count* is specified, it
114955 shall be applied to the motion command. A *count* shall be ignored for the following
114956 motion commands:

114957 `0 ^ $ c`

114958 If the motion command is the character `'c'`, the current command line shall be
114959 cleared and insert mode shall be entered. If the motion command would move the
114960 current cursor position toward the beginning of the command line, the character
114961 under the current cursor position shall not be deleted. If the motion command

114962 would move the current cursor position toward the end of the command line, the
114963 character under the current cursor position shall be deleted. If the *count* is larger
114964 than the number of characters between the current cursor position and the end of
114965 the command line toward which the motion command would move the cursor, this
114966 shall not be considered an error; all of the remaining characters in the
114967 aforementioned range shall be deleted and insert mode shall be entered. If the
114968 motion command is invalid, the terminal shall be alerted, the cursor shall not be
114969 moved, and no text shall be deleted.

114970 **C** Delete from the current character to the end of the line and enter insert mode at the
114971 new end-of-line.

114972 **S** Clear the entire edit line and enter insert mode.

114973 [*count*]**r***c* Replace the current character with the character `'c'`. With a number *count*,
114974 replace the current and the following *count*−1 characters. After this command, the
114975 current cursor position shall be on the last character that was changed. If the *count*
114976 is larger than the number of characters after the cursor, this shall not be considered
114977 an error; all of the remaining characters shall be changed.

114978 [*count*]**_** Append a <space> after the current character position and then append the last
114979 bigword in the previous input line after the <space>. Then enter insert mode after
114980 the last character just appended. With a number *count*, append the *count*th bigword
114981 in the previous line.

114982 [*count*]**x** Delete the character at the current cursor position and place the deleted characters
114983 in the save buffer. If the cursor was positioned on the last character of the line, the
114984 character shall be deleted and the cursor position shall be moved to the previous
114985 character (the new last character). If the *count* is larger than the number of
114986 characters after the cursor, this shall not be considered an error; all the characters
114987 from the cursor to the end of the line shall be deleted.

114988 [*count*]**X** Delete the character before the current cursor position and place the deleted
114989 characters in the save buffer. The character under the current cursor position shall
114990 not change. If the cursor was positioned on the first character of the line, the
114991 terminal shall be alerted, and the **X** command shall have no effect. If the line
114992 contained a single character, the **X** command shall have no effect. If the line
114993 contained no characters, the terminal shall be alerted and the cursor shall not be
114994 moved. If the *count* is larger than the number of characters before the cursor, this
114995 shall not be considered an error; all the characters from before the cursor to the
114996 beginning of the line shall be deleted.

114997 [*count*]**d***motion*
114998 Delete the characters between the current cursor position and the character
114999 position that would result from the motion command. A number *count* repeats the
115000 motion command *count* times. If the motion command would move toward the
115001 beginning of the command line, the character under the current cursor position
115002 shall not be deleted. If the motion command is **d**, the entire current command line
115003 shall be cleared. If the *count* is larger than the number of characters between the
115004 current cursor position and the end of the command line toward which the motion
115005 command would move the cursor, this shall not be considered an error; all of the
115006 remaining characters in the aforementioned range shall be deleted. The deleted
115007 characters shall be placed in the save buffer.

D Delete all characters from the current cursor position to the end of the line. The deleted characters shall be placed in the save buffer.

[*count*]**y***motion*
Yank (that is, copy) the characters from the current cursor position to the position resulting from the motion command into the save buffer. A number *count* shall be applied to the motion command. If the motion command would move toward the beginning of the command line, the character under the current cursor position shall not be included in the set of yanked characters. If the motion command is **y**, the entire current command line shall be yanked into the save buffer. The current cursor position shall be unchanged. If the *count* is larger than the number of characters between the current cursor position and the end of the command line toward which the motion command would move the cursor, this shall not be considered an error; all of the remaining characters in the aforementioned range shall be yanked.

Y Yank the characters from the current cursor position to the end of the line into the save buffer. The current character position shall be unchanged.

[*count*]**p** Put a copy of the current contents of the save buffer after the current cursor position. The current cursor position shall be advanced to the last character put from the save buffer. A *count* shall indicate how many copies of the save buffer shall be put.

[*count*]**P** Put a copy of the current contents of the save buffer before the current cursor position. The current cursor position shall be moved to the last character put from the save buffer. A *count* shall indicate how many copies of the save buffer shall be put.

u Undo the last command that changed the edit line. This operation shall not undo the copy of any command line to the edit line.

U Undo all changes made to the edit line. This operation shall not undo the copy of any command line to the edit line.

[*count*]**k**
[*count*]– Set the current command line to be the *count*th previous command line in the shell command history. If *count* is not specified, it shall default to 1. The cursor shall be positioned on the first character of the new command. If a **k** or – command would retreat past the maximum number of commands in effect for this shell (affected by the *HISTSIZE* environment variable), the terminal shall be alerted, and the command shall have no effect.

[*count*]**j**
[*count*]**+** Set the current command line to be the *count*th next command line in the shell command history. If *count* is not specified, it shall default to 1. The cursor shall be positioned on the first character of the new command. If a **j** or **+** command advances past the edit line, the current command line shall be restored to the edit line and the terminal shall be alerted.

[*number*]**G** Set the current command line to be the oldest command line stored in the shell command history. With a number *number*, set the current command line to be the command line *number* in the history. If command line *number* does not exist, the terminal shall be alerted and the command line shall not be changed.

115053 */pattern*<newline>

115054 Move backwards through the command history, searching for the specified
115055 pattern, beginning with the previous command line. Patterns use the pattern
115056 matching notation described in Section 2.14 (on page 2523), except that the '^'
115057 character shall have special meaning when it appears as the first character of
115058 *pattern*. In this case, the '^' is discarded and the characters after the '^' shall be
115059 matched only at the beginning of a line. Commands in the command history shall
115060 be treated as strings, not as filenames. If the pattern is not found, the current
115061 command line shall be unchanged and the terminal shall be alerted. If it is found in
115062 a previous line, the current command line shall be set to that line and the cursor
115063 shall be set to the first character of the new command line.

115064 If *pattern* is empty, the last non-empty pattern provided to **/** or **?** shall be used. If
115065 there is no previous non-empty pattern, the terminal shall be alerted and the
115066 current command line shall remain unchanged.

115067 **?***pattern*<newline>

115068 Move forwards through the command history, searching for the specified pattern,
115069 beginning with the next command line. Patterns use the pattern matching notation
115070 described in Section 2.14 (on page 2523), except that the '^' character shall have
115071 special meaning when it appears as the first character of *pattern*. In this case, the
115072 '^' is discarded and the characters after the '^' shall be matched only at the
115073 beginning of a line. Commands in the command history shall be treated as strings,
115074 not as filenames. If the pattern is not found, the current command line shall be
115075 unchanged and the terminal shall be alerted. If it is found in a following line, the
115076 current command line shall be set to that line and the cursor shall be set to the fist
115077 character of the new command line.

115078 If *pattern* is empty, the last non-empty pattern provided to **/** or **?** shall be used. If
115079 there is no previous non-empty pattern, the terminal shall be alerted and the
115080 current command line shall remain unchanged.

115081 **n** Repeat the most recent **/** or **?** command. If there is no previous **/** or **?**, the terminal
115082 shall be alerted and the current command line shall remain unchanged.

115083 **N** Repeat the most recent **/** or **?** command, reversing the direction of the search. If
115084 there is no previous **/** or **?**, the terminal shall be alerted and the current command
115085 line shall remain unchanged.

115086 **EXIT STATUS**

115087 The following exit values shall be returned:

115088 0 The script to be executed consisted solely of zero or more blank lines or comments, or
115089 both.

115090 1-125 A non-interactive shell detected an error other than *command_file* not found,
115091 *command_file* not executable, or an unrecoverable read error while reading commands
115092 (except from the *file* operand of the *dot* special built-in); including but not limited to
115093 syntax, redirection, or variable assignment errors.

115094 126 A specified *command_file* could not be executed due to an [ENOEXEC] error (see Section
115095 2.9.1.4 (on page 2502), item 2).

115096 127 A specified *command_file* could not be found by a non-interactive shell.

128 An unrecoverable read error was detected while reading commands, except from the *file* operand of the *dot* special built-in.

Otherwise, the shell shall terminate in the same manner as for an *exit* command with no operands, unless the last command the shell invoked was executed without forking, in which case the wait status seen by the parent process of the shell shall be the wait status of the last command the shell invoked. See the *exit* utility in Section 2.15 (on page 2526).

CONSEQUENCES OF ERRORS

See Section 2.8.1 (on page 2497).

APPLICATION USAGE

Standard input and standard error are the files that determine whether a shell is interactive when **–i** is not specified. For example:

```
sh > file
```

and:

```
sh 2> file
```

create interactive and non-interactive shells, respectively. Although both accept terminal input, the results of error conditions are different, as described in Section 2.8.1 (on page 2497); in the second example a redirection error encountered by a special built-in utility aborts the shell.

sh **–n** can be used to check for many syntax errors without waiting for *complete_commands* to be executed, but may be fooled into declaring false positives or missing actual errors that would occur when the shell actually evaluates *eval* commands present in the script, or if there are *alias* (or *unalias*) commands in the script that would alter the syntax of commands that use the affected aliases.

A conforming application must protect its first operand, if it starts with a <plus-sign>, by preceding it with the `"--"` argument that denotes the end of the options.

Applications should note that the standard *PATH* to the shell cannot be assumed to be either **/bin/sh** or **/usr/bin/sh**, and should be determined by interrogation of the *PATH* returned by *getconf PATH*, ensuring that the returned pathname is an absolute pathname and not a shell built-in.

For example, to determine the location of the standard *sh* utility:

```
command –v sh
```

On some implementations this might return:

```
/usr/xpg4/bin/sh
```

Furthermore, on systems that support executable scripts (the `"#!"` construct), it is recommended that applications using executable scripts install them using *getconf PATH* to determine the shell pathname and update the `"#!"` script appropriately as it is being installed (for example, with *sed*). For example:

```
#
# Installation time script to install correct POSIX shell pathname
#
# Get list of paths to check
#
Sifs=$IFS
Sifs_set=${IFS+y}
IFS=:
```



```
115141        set -- $(getconf PATH)
115142        if [ "$Sifs_set" = y ]
115143        then
115144            IFS=$Sifs
115145        else
115146            unset IFS
115147        fi
115148        #
115149        # Check each path for 'sh'
115150        #
115151        for i
115152        do
115153            if [ -x "${i}"/sh ]
115154            then
115155                Pshell=${i}/sh
115156            fi
115157        done
115158        #
115159        # This is the list of scripts to update. They should be of the
115160        # form '${name}.source' and will be transformed to '${name}'.
115161        # Each script should begin:
115162        #
115163        # #!INSTALLSHELLPATH
115164        #
115165        scripts="a b c"
115166        #
115167        # Transform each script
115168        #
115169        for i in ${scripts}
115170        do
115171            sed -e "s|INSTALLSHELLPATH|${Pshell}|" < ${i}.source > ${i}
115172        done
```

115173 EXAMPLES

115174 1. Execute a shell command from a string:

```
115175        sh -c "cat myfile"
```

115176 2. Execute a shell script from a file in the current directory:

```
115177        sh my_shell_cmds
```

115178 RATIONALE

115179 The *sh* utility and the *set* special built-in utility share a common set of options.

115180 The name *IFS* was originally an abbreviation of ``Input Field Separators''; however, this name is
115181 misleading as the *IFS* characters are actually used as field terminators. One justification for
115182 ignoring the contents of *IFS* upon entry to the script, beyond security considerations, is to assist
115183 possible future shell compilers. Allowing *IFS* to be imported from the environment prevents
115184 many optimizations that might otherwise be performed via dataflow analysis of the script itself.

115185 The text in the STDIN section about non-blocking reads concerns an instance of *sh* that has been
115186 invoked, probably by a C-language program, with standard input that has been opened using
115187 the O_NONBLOCK flag; see *open*( ) in the System Interfaces volume of POSIX.1-2024. If the shell
115188 did not reset this flag, it would immediately terminate because no input data would be available

yet and that would be considered the same as end-of-file.

The options associated with a *restricted shell* (command name *rsh* and the **–r** option) were excluded because the standard developers considered that the implied level of security could not be achieved and they did not want to raise false expectations.

On systems that support set-user-ID scripts, a historical trapdoor has been to link a script to the name **–i**. When it is called by a sequence such as:

```
sh -
```

or by:

```
#! usr/bin/sh -
```

the historical systems have assumed that no option letters follow. Thus, this volume of POSIX.1-2024 allows the single <hyphen-minus> to mark the end of the options, in addition to the use of the regular `"--"` argument, because it was considered that the older practice was so pervasive. An alternative approach is taken by the KornShell, where real and effective user/group IDs must match for an interactive shell; this behavior is specifically allowed by this volume of POSIX.1-2024.

Note: There are other problems with set-user-ID scripts that the two approaches described here do not resolve.

The initialization process for the history file can be dependent on the system start-up files, in that they may contain commands that effectively preempt the user's settings of *HISTFILE* and *HISTSIZE*. In some historical shells, the history file is initialized just after the *ENV* file has been processed. Therefore, it is implementation-defined whether changes made to *HISTFILE* after the history file has been initialized are effective.

The default messages for the various *MAIL*-related messages are unspecified because they vary across implementations. Typical messages are:

```
"you have mail\n"
```

or:

```
"you have new mail\n"
```

It is important that the descriptions of command line editing refer to the same shell as that in POSIX.1-2024 so that interactive users can also be application programmers without having to deal with programmatic differences in their two environments. It is also essential that the utility name *sh* be specified because this explicit utility name is too firmly rooted in historical practice of application programs for it to change.

Consideration was given to mandating a diagnostic message when attempting to set *vi*-mode on terminals that do not support command line editing. However, it is not historical practice for the shell to be cognizant of all terminal types and thus be able to detect inappropriate terminals in all cases. Implementations are encouraged to supply diagnostics in this case whenever possible, rather than leaving the user in a state where editing commands work incorrectly.

In early proposals, the KornShell-derived *emacs* mode of command line editing was included, even though the *emacs* editor itself was not. The community of *emacs* proponents was adamant that the full *emacs* editor not be standardized because they were concerned that an attempt to standardize this very powerful environment would encourage vendors to ship strictly conforming versions lacking the extensibility required by the community. The author of the original *emacs* program also expressed his desire to omit the program. Furthermore, there were a number of historical systems that did not include *emacs*, or included it without supporting it, but there were very few that did not include and support *vi*. The shell *emacs* command line editing

mode was finally omitted because it became apparent that the KornShell version and the editor being distributed with the GNU system had diverged in some respects. The author of *emacs* requested that the POSIX *emacs* mode either be deleted or have a significant number of unspecified conditions. Although the KornShell author agreed to consider changes to bring the shell into alignment, the standard developers decided to defer specification at that time. At the time, it was assumed that convergence on an acceptable definition would occur for a subsequent draft, but that has not happened, and there appears to be no impetus to do so. In any case, implementations are free to offer additional command line editing modes based on the exact models of editors their users are most comfortable with.

Early proposals had the following list entry in vi Line Editing Insert Mode (on page 3371):

\ If followed by the *erase* or *kill* character, that character shall be inserted into the input line. Otherwise, the <backslash> itself shall be inserted into the input line.

However, this is not actually a feature of *sh* command line editing insert mode, but one of some historical terminal line drivers. Some conforming implementations continue to do this when the *stty* **iexten** flag is set.

In interactive shells, SIGTERM is ignored so that `kill 0` does not kill the shell, and SIGINT is caught so that *wait* is interruptible. If the shell does not ignore SIGTTIN, SIGTTOU, and SIGTSTP signals when it is interactive and the **–m** option is not in effect, these signals suspend the shell if it is not a session leader. If it is a session leader, the signals are discarded if they would stop the process, as required by XSH Section 2.4.3 (on page 516) for orphaned process groups.

Earlier versions of this standard required that input files to the shell be text files except that line lengths were unlimited. However, that was overly restrictive in relation to the fact that shells can parse a script without a trailing newline, and in relation to a common practice of concatenating a shell script ending with an `exit` or `exec $command` with a binary data payload to form a single-file self-extracting archive.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

Section 2.9.1.4 (on page 2502), Chapter 2 (on page 2472), *cd*, *echo*, *exit*, *fc*, *pwd*, invalid, *set*, *stty*, *test*, *trap*, *umask*, *vi*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *dup*( ), *exec*, *exit*( ), *fork*( ), *getrlimit*( ), *open*( ), *pipe*( ), *signal*( ), *system*( ), *umask*( ), *wait*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Text is added to the DESCRIPTION for the Large File Summit proposal.

Issue 6

The Open Group Corrigendum U029/2 is applied, correcting the second SYNOPSIS.

The Open Group Corrigendum U027/3 is applied, correcting a typographical error.

The following new requirements on POSIX implementations derive from alignment with the

Single UNIX Specification:

- The option letters derived from the *set* special built-in are also accepted with a leading <plus-sign> (`'+'`).
- Large file extensions are added:
  - Pathname expansion does not fail due to the size of a file.
  - Shell input and output redirections have an implementation-defined offset maximum that is established in the open file description.

In the ENVIRONMENT VARIABLES section, the text ``user's home directory'' is updated to ``directory referred to by the *HOME* environment variable''.

Descriptions for the *ENV* and *PWD* environment variables are included to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #098 is applied, changing the definition of *IFS*.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Changes to the *pwd* utility and *PWD* environment variable have been made to match the changes to the *getcwd*( ) function made for Austin Group Interpretation 1003.1-2001 #140.

Minor editorial changes are made to the User Portability Utilities option shading. No normative changes are implied.

Minor changes are made to the install script example in the APPLICATION USAGE section.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0137 [152], XCU/TC1-2008/0138 [347], XCU/TC1-2008/0139 [347], XCU/TC1-2008/0140 [347], XCU/TC1-2008/0141 [299], and XCU/TC1-2008/0142 [347] are applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0175 [584], XCU/TC2-2008/0176 [584], XCU/TC2-2008/0177 [718], XCU/TC2-2008/0178 [884], XCU/TC2-2008/0179 [809], XCU/TC2-2008/0180 [884], and XCU/TC2-2008/0181 [584] are applied.

Issue 8

Austin Group Defect 51 is applied, changing the EXIT STATUS section.

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 981 is applied, removing a reference to the *set* **−o** *nolog* option from the RATIONALE section.

Austin Group Defect 1006 is applied, changing the description of the *ENV* environment variable.

Austin Group Defect 1055 is applied, adding a paragraph about the **−n** option to the APPLICATION USAGE section.

Austin Group Defect 1063 is applied, adding OB shading to the **−h** option and adding it to the list of options that are described as part of the *set* utility.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1250 is applied, changing the INPUT FILES section.

Austin Group Defect 1266 is applied, clarifying the circumstances under which the shell is

115318 considered to be interactive.

115319 Austin Group Defect 1267 is applied, changing the ENVIRONMENT VARIABLES section to
115320 remove the UP shading from *HOME* and add it to *HISTSIZE*.

115321 Austin Group Defect 1519 is applied, making the behavior explicitly unspecified if the **−o** or **+o**
115322 option is specified without an option-argument.

115323 Austin Group Defect 1629 is applied, changing the EXIT STATUS section.

NAME

sleep — suspend execution for an interval

SYNOPSIS

```
sleep time
```

DESCRIPTION

The *sleep* utility shall suspend execution for at least the integral number of seconds specified by the *time* operand.

OPTIONS

None.

OPERANDS

The following operand shall be supported:

time A non-negative decimal integer specifying the number of seconds for which to suspend execution.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *sleep*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

If the *sleep* utility receives a SIGALRM signal, one of the following actions shall be taken:

1. Terminate normally with a zero exit status.
2. Effectively ignore the signal.
3. Provide the default behavior for signals described in the ASYNCHRONOUS EVENTS section of Section 1.4 (on page 2462). This could include terminating with a non-zero exit status.

The *sleep* utility shall take the standard action for all other signals.

115363 **STDOUT**

115364 Not used.

115365 **STDERR**

115366 The standard error shall be used only for diagnostic messages.

115367 **OUTPUT FILES**

115368 None.

115369 **EXTENDED DESCRIPTION**

115370 None.

115371 **EXIT STATUS**

115372 The following exit values shall be returned:

115373 0 The execution was successfully suspended for at least *time* seconds, or a SIGALRM signal
115374 was received. See the ASYNCHRONOUS EVENTS section.

115375 >0 An error occurred.

115376 **CONSEQUENCES OF ERRORS**

115377 Default.

115378 **APPLICATION USAGE**

115379 None.

115380 **EXAMPLES**

115381 The *sleep* utility can be used to execute a command after a certain amount of time, as in:

115382 ```
(sleep 105; command) &
```

115383 or to execute a command every so often, as in:

115384 ```
while true
115385 do
115386     command
115387     sleep 37
115388 done
```

115389 **RATIONALE**

115390 The exit status is allowed to be zero when *sleep* is interrupted by the SIGALRM signal because
115391 most implementations of this utility rely on the arrival of that signal to notify them that the
115392 requested finishing time has been successfully attained. Such implementations thus do not
115393 distinguish this situation from the successful completion case. Other implementations are
115394 allowed to catch the signal and go back to sleep until the requested time expires or to provide
115395 the normal signal termination procedures.

115396 As with all other utilities that take integral operands and do not specify subranges of allowed
115397 values, *sleep* is required by this volume of POSIX.1-2024 to deal with *time* requests of up to
115398 2 147 483 647 seconds. This may mean that some implementations have to make multiple calls to
115399 the delay mechanism of the underlying operating system if its argument range is less than this.

115400 **FUTURE DIRECTIONS**

115401 None.

115402 **SEE ALSO**

115403 *wait*

115404 XBD Chapter 8 (on page 167)

115405 XSH *alarm*( ), *sleep*( )

115406 **CHANGE HISTORY**

115407 First released in Issue 2.

115408 **Issue 8**

115409 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

sort — sort, merge, or sequence check text files

SYNOPSIS

```
sort [-m] [-o output] [-bdfinru] [-t char] [-k keydef]... [file...]
sort [-c|-C] [-bdfinru] [-t char] [-k keydef] [file]
```

DESCRIPTION

The *sort* utility shall perform one of the following functions:

1. Sort lines of all the named files together and write the result to the specified output.
2. Merge lines of all the named (presorted) files together and write the result to the specified output.
3. Check that a single input file is correctly presorted.

Comparisons shall be based on one or more sort keys extracted from each line of input (or, if no sort keys are specified, the entire line up to, but not including, the terminating <newline>), and shall be performed using the collating sequence of the current locale. If this collating sequence does not have a total ordering of all characters (see XBD Section 7.3.2, on page 139), any lines of input that collate equally shall be further compared byte-by-byte using the collating sequence for the POSIX locale.

OPTIONS

The *sort* utility shall conform to XBD Section 12.2 (on page 215), except for Guideline 9, and the **–k** *keydef* option should follow the **–b**, **–d**, **–f**, **–i**, **–n**, and **–r** options. In addition, `'+'` may be recognized as an option delimiter as well as `'–'`.

The following options shall be supported:

–c Check that the single input file is ordered as specified by the arguments and the collating sequence of the current locale. Output shall not be sent to standard output. The exit code shall indicate whether or not disorder was detected or an error occurred. If disorder (or, with **–u**, a duplicate key) is detected, a warning message shall be sent to standard error indicating where the disorder or duplicate key was found.

–C Same as **–c**, except that a warning message shall not be sent to standard error if disorder or, with **–u**, a duplicate key is detected.

–m Merge only; the input file shall be assumed to be already sorted.

–o *output* Specify the name of an output file to be used instead of the standard output. This file can be the same as one of the input *file*s.

–u Unique: suppress all but one in each set of lines having equal keys. If used with the **–c** option, check that there are no lines with duplicate keys, in addition to checking that the input file is sorted.

The following options shall override the default ordering rules. When ordering options appear independent of any key field specifications, the requested field ordering rules shall be applied globally to all sort keys. When attached to a specific key (see **–k**), the specified ordering options shall override all global ordering options for that key.

–d Specify that only <blank> characters and alphanumeric characters, according to the current setting of *LC_CTYPE*, shall be significant in comparisons. The behavior is undefined for a sort key to which **–i** or **–n** also applies.

115453 **−f** Consider all lowercase characters that have uppercase equivalents, according to
115454 the current setting of *LC_CTYPE*, to be the uppercase equivalent for the purposes
115455 of comparison.

115456 **−i** Ignore all characters that are non-printable, according to the current setting of
115457 *LC_CTYPE*. The behavior is undefined for a sort key for which **−n** also applies.

115458 **−n** Restrict the sort key to an initial numeric string, consisting of optional <blank>
115459 characters, optional <hyphen-minus> character, and zero or more digits with an
115460 optional radix character and thousands separators (as defined in the current
115461 locale), which shall be sorted by arithmetic value. An empty digit string shall be
115462 treated as zero. Leading zeros and signs on zeros shall not affect ordering.

115463 **−r** Reverse the sense of comparisons.

115464 The treatment of field separators can be altered using the options:

115465 **−b** Ignore leading <blank> characters when determining the starting and ending
115466 positions of a restricted sort key. If the **−b** option is specified before the first **−k**
115467 option, it shall be applied to all **−k** options. Otherwise, the **−b** option can be
115468 attached independently to each **−k** *field_start* or *field_end* option-argument (see
115469 below).

115470 **−t** *char* Use *char* as the field separator character; *char* shall not be considered to be part of a
115471 field (although it can be included in a sort key). Each occurrence of *char* shall be
115472 significant (for example, <*char*><*char*> delimits an empty field). If **−t** is not
115473 specified, <blank> characters shall be used as default field separators; each
115474 maximal non-empty sequence of <blank> characters that follows a non-<blank>
115475 shall be a field separator.

115476 Sort keys can be specified using the options:

115477 **−k** *keydef* The *keydef* argument is a restricted sort key field definition. The format of this
115478 definition is:

115479 *field_start*`[`*type*`][,`*field_end*`[`*type*`]]`

115480 where *field_start* and *field_end* define a key field restricted to a portion of the line
115481 (see the EXTENDED DESCRIPTION section), and *type* is one or more modifiers
115482 from the list of characters `'b'`, `'d'`, `'f'`, `'i'`, `'n'`, `'r'`. The `'b'` modifier shall
115483 behave like the **−b** option, but shall apply only to the *field_start* or *field_end* to
115484 which it is attached. The other modifiers shall behave like the corresponding
115485 options, but shall apply only to the key field to which they are attached; they shall
115486 have this effect if specified with *field_start*, *field_end*, or both. If any modifier is
115487 attached to a *field_start* or to a *field_end*, no option shall apply to either.
115488 Implementations shall support at least nine occurrences of the **−k** option, which
115489 shall be significant in command line order. If no **−k** option is specified, a default
115490 sort key of the entire line shall be used.

115491 When there are multiple key fields, later keys shall be compared only after all
115492 earlier keys compare equal. Except when the **−u** option is specified, lines that
115493 otherwise compare equal shall be ordered as if none of the options **−d**, **−f**, **−i**, **−n**, or
115494 **−k** were present (but with **−r** still in effect, if it was specified) and with all bytes in
115495 the lines significant to the comparison. The order in which lines that still compare
115496 equal are written is unspecified.

OPERANDS

The following operand shall be supported:

file A pathname of a file to be sorted, merged, or checked. If no *file* operands are specified, or if a *file* operand is `'-'`, the standard input shall be used. If *sort* encounters an error when opening or reading a *file* operand, it may exit without writing any output to standard output or processing later operands.

STDIN

The standard input shall be used only if no *file* operands are specified, or if a *file* operand is `'-'`. See the INPUT FILES section.

INPUT FILES

The input files shall be text files, except that the *sort* utility shall add a <newline> to the end of a file ending with an incomplete last line.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *sort*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for ordering rules.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classification for the **−b**, **−d**, **−f**, **−i**, and **−n** options.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

LC_NUMERIC
Determine the locale for the definition of the radix character and thousands separator for the **−n** option.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

TMPDIR Provide a pathname that shall override the default directory for temporary files, if any.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Unless the **−o** or **−c** options are in effect, the standard output shall contain the sorted input.

STDERR

The standard error shall be used for diagnostic messages. When **−c** is specified, if disorder is detected (or if **−u** is also specified and a duplicate key is detected), a message shall be written to the standard error which identifies the input line at which disorder (or a duplicate key) was detected. A warning message about correcting an incomplete last line of an input file may be generated, but need not affect the final exit status.

OUTPUT FILES

If the −**o** option is in effect, the sorted input shall be written to the file *output*.

EXTENDED DESCRIPTION

The notation:

```
-k field_start[type][,field_end[type]]
```

shall define a key field that begins at *field_start* and ends at *field_end* inclusive, unless *field_start* falls beyond the end of the line or after *field_end*, in which case the key field is empty. A missing *field_end* shall mean the last character of the line.

A field comprises a maximal sequence of non-separating characters and, in the absence of option −**t**, any preceding field separator.

The *field_start* portion of the *keydef* option-argument shall have the form:

```
field_number[.first_character]
```

Fields and characters within fields shall be numbered starting with 1. The *field_number* and *first_character* pieces, interpreted as positive decimal integers, shall specify the first character to be used as part of a sort key. If *.first_character* is omitted, it shall refer to the first character of the field.

The *field_end* portion of the *keydef* option-argument shall have the form:

```
field_number[.last_character]
```

The *field_number* shall be as described above for *field_start*. The *last_character* piece, interpreted as a non-negative decimal integer, shall specify the last character to be used as part of the sort key. If *last_character* evaluates to zero or *.last_character* is omitted, it shall refer to the last character of the field specified by *field_number*.

If the −**b** option or **b** type modifier is in effect, characters within a field shall be counted from the first non-<blank> in the field. (This shall apply separately to *first_character* and *last_character*.)

EXIT STATUS

The following exit values shall be returned:

0 All input files were output successfully, or −**c** was specified and the input file was correctly sorted.

1 Under the −**c** option, the file was not ordered as specified, or if the −**c** and −**u** options were both specified, two input lines were found with equal keys.

>1 An error occurred.

CONSEQUENCES OF ERRORS

The default requirements shall apply, except that if *sort* encounters an error when opening or reading a *file* operand, it may exit without writing any output to standard output or processing later operands.

APPLICATION USAGE

The default value for **–t**, <blank>, has different properties from, for example, **–t**"<space>". If a line contains:

```
<space><space>foo
```

the following treatment would occur with default separation as opposed to specifically selecting a <space>:

Field	Default	–t "<space>"
1	<space><space>foo	*empty*
2	*empty*	*empty*
3	*empty*	foo

The leading field separator itself is included in a field when **–t** is not used. For example, this command returns an exit status of zero, meaning the input was already sorted:

```
sort –c –k 2 <<eof
y<tab>b
x<space>a
eof
```

(assuming that a <tab> precedes the <space> in the current collating sequence). The field separator is not included in a field when it is explicitly set via **–t**. This is historical practice and allows usage such as:

```
sort –t "|" –k 2n <<eof
Atlanta|425022|Georgia
Birmingham|284413|Alabama
Columbia|100385|South Carolina
eof
```

where the second field can be correctly sorted numerically without regard to the non-numeric field separator.

The wording in the OPTIONS section clarifies that the **–b**, **–d**, **–f**, **–i**, **–n**, and **–r** options have to come before the first sort key specified if they are intended to apply to all specified keys. The way it is described in this volume of POSIX.1-2024 matches historical practice, not historical documentation. The results are unspecified if these options are specified after a **–k** option.

The **–f** option might not work as expected in locales where there is not a one-to-one mapping between an uppercase and a lowercase letter.

When using *sort* to process pathnames, it is recommended that LC_ALL, or at least LC_CTYPE and LC_COLLATE, are set to POSIX or C in the environment, since pathnames can contain byte sequences that do not form valid characters in some locales, in which case the utility's behavior would be undefined. In the POSIX locale each byte is a valid single-byte character, and therefore this problem is avoided.

If the collating sequence of the current locale does not have a total ordering of all characters, since *sort* **–u** suppresses lines with duplicate keys, it suppresses lines that collate equally but are not identical.

EXAMPLES

1. The following command sorts the contents of **infile** with the second field as the sort key:

   ```
   sort –k 2,2 infile
   ```


2. The following command sorts, in reverse order, the contents of **infile1** and **infile2**, placing the output in **outfile** and using the second character of the second field as the sort key (assuming that the first character of the second field is the field separator):

```
sort -r -o outfile -k 2.2,2.2 infile1 infile2
```

3. The following command sorts the contents of **infile1** and **infile2** using the second non-<blank> of the second field as the sort key:

```
sort -k 2.2b,2.2b infile1 infile2
```

4. The following command prints the System V password file (user database) sorted by the numeric user ID (the third <colon>-separated field):

```
sort -t : -k 3,3n /etc/passwd
```

5. The following command prints the lines of the already sorted file **infile**, suppressing all but one occurrence of lines having the same third field:

```
sort -um -k 3.1,3.0 infile
```

RATIONALE

Examples in some historical documentation state that options –**um** with one input file keep the first in each set of lines with equal keys. This behavior was deemed to be an implementation artifact and was not standardized.

The –**z** option was omitted; it is not standard practice on most systems and is inconsistent with using *sort* to sort several files individually and then merge them together. The text concerning –**z** in historical documentation appeared to require implementations to determine the proper buffer length during the sort phase of operation, but not during the merge.

The –**y** option was omitted because of non-portability. The –**M** option, present in System V, was omitted because of non-portability in international usage.

An undocumented –**T** option exists in some implementations. It is used to specify a directory for intermediate files. Implementations are encouraged to support the use of the *TMPDIR* environment variable instead of adding an option to support this functionality.

The –**k** option was added to satisfy two objections. First, the zero-based counting used by *sort* is not consistent with other utility conventions. Second, it did not meet syntax guideline requirements.

Historical documentation indicates that ``setting –**n** implies –**b**''. The description of –**n** already states that optional leading <blank>s are tolerated in doing the comparison. If –**b** is enabled, rather than implied, by –**n**, this has unusual side-effects. When a character offset is used in a column of numbers (for example, to sort modulo 100), that offset is measured relative to the most significant digit, not to the column. Based upon a recommendation from the author of the original *sort* utility, the –**b** implication has been omitted from this volume of POSIX.1-2024, and an application wishing to achieve the previously mentioned side-effects has to code the –**b** flag explicitly.

Earlier versions of this standard allowed the –**o** option to appear after operands. Historical practice allowed all options to be interspersed with operands. This version of the standard allows implementations to accept options after operands but conforming applications should not use this form.

Earlier versions of this standard also allowed the *–number* and *+number* options. These options are no longer specified by POSIX.1-2024 but may be present in some implementations.

Historical implementations produced a message on standard error when –**c** was specified and

115663 disorder was detected, and when **–c** and **–u** were specified and a duplicate key was detected. An
115664 earlier version of this standard contained wording that did not make it clear that this message
115665 was allowed and some implementations removed this message to be sure that they conformed
115666 to the standard's requirements. Confronted with this difference in behavior, interactive users
115667 that wanted to be sure that they got visual feedback instead of just exit code 1 could have used a
115668 command like:

115669 `sort -c file || echo disorder`

115670 whether or not the *sort* utility provided a message in this case. But, it was not easy for a user to
115671 find where the disorder or duplicate key occurred on implementations that do not produce a
115672 message, especially when some parts of the input line were not part of the key and when one or
115673 more of the **–b**, **–d**, **–f**, **–i**, **–n**, or –r options or *keydef* type modifiers were in use. POSIX.1-2024
115674 requires a message to be produced in this case. POSIX.1-2024 also contains the **–C** option giving
115675 users the ability to choose either behavior.

115676 When a disorder or duplicate is found when the **–c** option is specified, some implementations
115677 print a message containing the first line that is out of order or contains a duplicate key; others
115678 print a message specifying the line number of the offending line. This standard allows either
115679 type of message.

115680 The required further byte-by-byte comparison of lines that collate equally may have an impact
115681 on efficiency, but this can be mitigated by only performing the additional comparison if the
115682 current locale's collating sequence does not have a total ordering of all characters (if the
115683 implementation provides a way to query this) or by only performing the additional comparison
115684 if the locale name associated with the *LC_COLLATE* category has an `'@'` modifier in the name
115685 (since implementation-supplied locales without an `'@'` modifier have a total ordering of all
115686 characters — see XBD Section 7.3.2 (on page 139) — and *localedef* users are warned to follow the
115687 same convention). Note that if the implementation provides a *stable sort* option as an extension
115688 (usually **–s**), the additional comparison should not be performed when this option has been
115689 specified.

115690 **FUTURE DIRECTIONS**

115691 If this utility is directed to create a new directory entry that contains any bytes that have the
115692 encoded value of a <newline> character, implementations are encouraged to treat this as an
115693 error. A future version of this standard may require implementations to treat this as an error.

115694 **SEE ALSO**

115695 *comm*, *join*, *uniq*

115696 XBD Section 7.3.2 (on page 139), Chapter 8 (on page 167), Section 12.2 (on page 215)

115697 XSH *toupper*( )

115698 **CHANGE HISTORY**

115699 First released in Issue 2.

115700 **Issue 6**

115701 IEEE PASC Interpretation 1003.2 #174 is applied, updating the DESCRIPTION of comparisons.

115702 IEEE PASC Interpretation 1003.2 #168 is applied.

115703 **Issue 7**

115704 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that Guideline 9 of the Utility
115705 Syntax Guidelines does not apply and noting that `'+'` may be recognized as an option delimiter.

115706 Austin Group Interpretation 1003.1-2001 #120 is applied, clarifying the use of the **–c** option and
115707 introducing the **–C** option.

XCU-ERN-81 is applied, modifying the description of the **–i** option.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0182 [963], XCU/TC2-2008/0183 [584], XCU/TC2-2008/0184 [510], XCU/TC2-2008/0185 [962], XCU/TC2-2008/0186 [663], and XCU/TC2-2008/0187 [963] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 862 is applied, adding *TMPDIR* to the ENVIRONMENT VARIABLES section.

Austin Group Defect 1070 is applied, requiring that any lines of input that collate equally when comparing them as whole lines are further compared byte-by-byte using the collating sequence for the POSIX locale.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

split — split a file into pieces

SYNOPSIS

```
split [-l line_count] [-a suffix_length] [file [name]]
split -b n[k|m] [-a suffix_length] [file [name]]
```

DESCRIPTION

The *split* utility shall read an input file and write zero or more output files. The default size of each output file shall be 1 000 lines. The size of the output files can be modified by specification of the **–b** or **–l** options. Each output file shall be created with a unique suffix. The suffix shall consist of exactly *suffix_length* lowercase letters from the POSIX locale. The letters of the suffix shall be used as if they were a base-26 digit system, with the first suffix to be created consisting of all `'a'` characters, the second with a `'b'` replacing the last `'a'`, and so on, until a name of all `'z'` characters is created. By default, the names of the output files shall be `'x'`, followed by a two-character suffix from the character set as described above, starting with `"aa"`, `"ab"`, `"ac"`, and so on, and continuing until the suffix `"zz"`, for a maximum of 676 files.

If the number of files required exceeds the maximum allowed by the suffix length provided, such that the last allowable file would be larger than the requested size, the *split* utility shall fail after creating the last file with a valid suffix; *split* shall not delete the files it created with valid suffixes. If the file limit is not exceeded, the last file created shall contain the remainder of the input file, and may be smaller than the requested size. If the input is an empty file, no output file shall be created and this shall not be considered to be an error.

OPTIONS

The *split* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–a *suffix_length*
: Use *suffix_length* letters to form the suffix portion of the filenames of the split file. If **–a** is not specified, the default suffix length shall be two. If the sum of the *name* operand and the *suffix_length* option-argument would create a filename exceeding {NAME_MAX} bytes, an error shall result; *split* shall exit with a diagnostic message and no files shall be created.

–b *n*
: Split a file into pieces *n* bytes in size.

–b *n***k**
: Split a file into pieces *n**1 024 bytes in size.

–b *n***m**
: Split a file into pieces *n**1 048 576 bytes in size.

–l *line_count*
: Specify the number of lines in each resulting file piece. The *line_count* argument is an unsigned decimal integer. The default is 1 000. If the input does not end with a <newline>, the partial line shall be included in the last output file.

OPERANDS

The following operands shall be supported:

file
: The pathname of the ordinary file to be split. If no input file is given or *file* is `'–'`, the standard input shall be used.

name
: The prefix to be used for each of the files resulting from the split operation. If no *name* argument is given, `'x'` shall be used as the prefix of the output files. The combined length of the basename of *prefix* and *suffix_length* cannot exceed {NAME_MAX} bytes. See the OPTIONS section.

STDIN

See the INPUT FILES section.

INPUT FILES

Any file can be used as input.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *split*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

The output files contain portions of the original input file; otherwise, unchanged.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

In the following examples **foo** is a text file that contains 5 000 lines.

1. Create five files, **xaa**, **xab**, **xac**, **xad**, and **xae**:

```
split foo
```

2. Create five files, but the suffixed portion of the created files consists of three letters, **xaaa**, **xaab**, **xaac**, **xaad**, and **xaae**:

```
split -a 3 foo
```

3. Create three files with four-letter suffixes and a supplied prefix, **bar_aaaa**, **bar_aaab**, and **bar_aaac**:

```
split -a 4 -l 2000 foo bar_
```

4. Create as many files as are necessary to contain at most 20*1 024 bytes, each with the default prefix of **x** and a five-letter suffix:

```
split -a 5 -b 20k foo
```

RATIONALE

The **-b** option was added to provide a mechanism for splitting files other than by lines. While most uses of the **-b** option are for transmitting files over networks, some believed it would have additional uses.

The **-a** option was added to overcome the limitation of being able to create only 676 files.

Consideration was given to deleting this utility, using the rationale that the functionality provided by this utility is available via the *csplit* utility (see *csplit*). Upon reconsideration of the purpose of the User Portability Utilities option, it was decided to retain both this utility and the *csplit* utility because users use both utilities and have historical expectations of their behavior. Furthermore, the splitting on byte boundaries in *split* cannot be duplicated with the historical *csplit*.

The text ``*split* shall not delete the files it created with valid suffixes'' would normally be assumed, but since the related utility, *csplit*, does delete files under some circumstances, the historical behavior of *split* is made explicit to avoid misinterpretation.

Earlier versions of this standard allowed a *-line_count* option. This form is no longer specified by POSIX.1-2024 but may be present in some implementations.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

csplit

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The APPLICATION USAGE section is added.

The obsolescent SYNOPSIS is removed.

Issue 7

Austin Group Interpretation 1003.1-2001 #027 is applied.

The *split* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0188 [731] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

strings — find printable strings in files

SYNOPSIS

```
strings [-a] [-t format] [-n number] [file...]
```

DESCRIPTION

The *strings* utility shall look for printable strings in regular files and shall write those strings to standard output. A printable string is any sequence of four (by default) or more printable characters terminated by a <newline> or NUL character. Additional implementation-defined strings may be written; see *localedef*.

If any argument is `'-'`, the results are unspecified.

OPTIONS

The *strings* utility shall conform to XBD Section 12.2 (on page 215), except for the unspecified usage of `'-'`.

The following options shall be supported:

-a Scan files in their entirety. If **-a** is not specified, it is implementation-defined what portion of each file is scanned for strings.

-n *number* Specify the minimum string length, where the *number* argument is a positive decimal integer. The default shall be 4.

-t *format* Write each string preceded by its byte offset from the start of the file. The format shall be dependent on the single character used as the *format* option-argument:

- `d` The offset shall be written in decimal.
- `o` The offset shall be written in octal.
- `x` The offset shall be written in hexadecimal.

OPERANDS

The following operand shall be supported:

file A pathname of a regular file to be used as input. If no *file* operand is specified, the *strings* utility shall read from the standard input.

STDIN

See the INPUT FILES section.

INPUT FILES

The input files named by the utility arguments or the standard input shall be regular files of any format.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *strings*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and to identify printable strings.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Strings found shall be written to the standard output, one per line.

When the **–t** option is not specified, the format of the output shall be:

`"%s", <string>`

With the **–t o** option, the format of the output shall be:

`"%o %s", <byte offset>, <string>`

With the **–t x** option, the format of the output shall be:

`"%x %s", <byte offset>, <string>`

With the **–t d** option, the format of the output shall be:

`"%d %s", <byte offset>, <string>`

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

By default the data area (as opposed to the text, ``bss'', or header areas) of a binary executable file is scanned. Implementations document which areas are scanned.

Some historical implementations do not require NUL or <newline> terminators for strings to permit those languages that do not use NUL as a string terminator to have their strings written.

EXAMPLES

None.

RATIONALE

Apart from rationalizing the option syntax and slight difficulties with object and executable binary files, *strings* is specified to match historical practice closely. The **–a** and **–n** options were introduced to replace the non-conforming – and *–number* options. These options are no longer specified by POSIX.1-2024 but may be present in some implementations.

The **–o** option historically means different things on different implementations. Some use it to mean ``*offset* in decimal'', while others use it as ``*offset* in octal''. Instead of trying to decide which way would be least objectionable, the **–t** option was added. It was originally named **–O** to mean ``offset'', but was changed to **–t** to be consistent with *od*.

The ISO C standard function *isprint*( ) is restricted to a domain of **unsigned char**. This volume of POSIX.1-2024 requires implementations to write strings as defined by the current locale.

FUTURE DIRECTIONS

None.

SEE ALSO

localedef, *nm*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The obsolescent SYNOPSIS is removed.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first argument is `'–'`.

The *strings* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1599 is applied, making the behavior unspecified when any argument is `'–'`.

NAME

strip — remove unnecessary information from strippable files (**DEVELOPMENT**)

SYNOPSIS

SD
```
strip file...
```

DESCRIPTION

XSI A strippable file is defined as a relocatable, object, or executable file. On XSI-conformant systems, a strippable file can also be an archive of object or relocatable files.

The *strip* utility shall remove from strippable files named by the *file* operands any information the implementor deems unnecessary for execution of those files. The nature of that information is unspecified. The effect of *strip* on object and executable files shall be similar to the use of the XSI **−s** option to *c17*. The effect of *strip* on an archive of object files shall be similar to the use of the **−s** option to *c17* for each object file in the archive.

OPTIONS

None.

OPERANDS

The following operand shall be supported:

file A pathname referring to a strippable file.

STDIN

Not used.

INPUT FILES

The input files shall be in the form of strippable files successfully produced by any compiler XSI defined by this volume of POSIX.1-2024 or produced by creating or updating an archive of such files using the *ar* utility.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *strip*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

116004 **STDERR**

116005 The standard error shall be used only for diagnostic messages.

116006 **OUTPUT FILES**

116007 The *strip* utility shall produce strippable files of unspecified format.

116008 **EXTENDED DESCRIPTION**

116009 None.

116010 **EXIT STATUS**

116011 The following exit values shall be returned:

116012 0 Successful completion.

116013 >0 An error occurred.

116014 **CONSEQUENCES OF ERRORS**

116015 Default.

116016 **APPLICATION USAGE**

116017 None.

116018 **EXAMPLES**

116019 None.

116020 **RATIONALE**

116021 Historically, this utility has been used to remove the symbol table from a strippable file. It was
116022 included since it is known that the amount of symbolic information can amount to several
116023 megabytes; the ability to remove it in a portable manner was deemed important, especially for
116024 smaller systems.

116025 The behavior of *strip* on object and executable files is said to be the same as the **–s** option to a
116026 compiler. While the end result is essentially the same, it is not required to be identical.

116027 XSI-conformant systems support use of *strip* on archive files containing object files or relocatable
116028 files.

116029 **FUTURE DIRECTIONS**

116030 None.

116031 **SEE ALSO**

116032 *ar*, *c17*

116033 XBD Chapter 8 (on page 167)

116034 **CHANGE HISTORY**

116035 First released in Issue 2.

116036 **Issue 6**

116037 This utility is marked as part of the Software Development Utilities option.

116038 **Issue 7**

116039 Austin Group Interpretation 1003.1-2001 #103 is applied.

116040 **Issue 8**

116041 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

116042 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

NAME

stty — set the options for a terminal

SYNOPSIS

```
stty [-a|-g]
stty operand...
```

DESCRIPTION

The *stty* utility shall set or report on terminal I/O characteristics for the device that is its standard input. Without options or operands specified, it shall report the settings of certain characteristics, usually those that differ from implementation-defined defaults. Otherwise, it shall modify the terminal state according to the specified operands. Detailed information about the modes listed in the first five groups below are described in XBD Chapter 11 (on page 199). Operands in the Combination Modes group (see Combination Modes, on page 3410) are implemented using operands in the previous groups. Some combinations of operands are mutually-exclusive on some terminal types; the results of using such combinations are unspecified.

Typical implementations of this utility require a communications line configured to use the **termios** interface defined in the System Interfaces volume of POSIX.1-2024. On systems where none of these lines are available, and on lines not currently configured to support the **termios** interface, some of the operands need not affect terminal characteristics.

OPTIONS

The *stty* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−a Write to standard output all the current settings for the terminal.

−g Write to standard output all the current settings, optionally excluding the terminal window size, in an unspecified form that, when stripped of trailing <newline> characters, and used as the one and only argument to another invocation of the *stty* utility on the same system, attempts to apply those settings to the terminal. The form used shall not contain any sequence that would form an Informational Query, and shall consist of one line of text consisting of only printable characters from the portable character set, excluding white-space characters (other than the terminating <newline>) and these characters that could be altered by pathname expansion performed by the shell: `'*'`, `'?'`, and `'['`.

OPERANDS

The following operands shall be supported.

Control Modes

parenb (**−parenb**) Enable (disable) parity generation and detection. This shall have the effect of setting (not setting) PARENB in the **termios** *c_cflag* field, as defined in XBD Chapter 11 (on page 199).

parodd (**−parodd**)
Select odd (even) parity. This shall have the effect of setting (not setting) PARODD in the **termios** *c_cflag* field, as defined in XBD Chapter 11 (on page 199).

cs5 cs6 cs7 cs8 Select character size, if possible. This shall have the effect of setting CS5, CS6, CS7, and CS8, respectively, in the **termios** *c_cflag* field, as defined in XBD Chapter 11 (on page 199).

number Set terminal baud rate to the number given, if possible. If the baud rate is set to zero, the modem control lines shall no longer be asserted. This shall have the effect of setting the input and output **termios** baud rate values as defined in XBD Chapter 11 (on page 199).

ispeed *number* Set terminal input baud rate to the number given, if possible. If the input baud rate is set to zero, the input baud rate shall be specified by the value of the output baud rate. This shall have the effect of setting the input **termios** baud rate value as defined in XBD Chapter 11 (on page 199).

ospeed *number* Set terminal output baud rate to the number given, if possible. If the output baud rate is set to zero, the modem control lines shall no longer be asserted. This shall have the effect of setting the output **termios** baud rate value as defined in XBD Chapter 11 (on page 199).

hupcl (**–hupcl**) Stop asserting modem control lines (do not stop asserting modem control lines) on last close. This shall have the effect of setting (not setting) HUPCL in the **termios** *c_cflag* field, as defined in XBD Chapter 11 (on page 199).

hup (**–hup**) Equivalent to **hupcl**(**–hupcl**).

cstopb (**–cstopb**) Use two (one) stop bits per character. This shall have the effect of setting (not setting) CSTOPB in the **termios** *c_cflag* field, as defined in XBD Chapter 11 (on page 199).

cread (**–cread**) Enable (disable) the receiver. This shall have the effect of setting (not setting) CREAD in the **termios** *c_cflag* field, as defined in XBD Chapter 11 (on page 199).

clocal (**–clocal**) Assume a line without (with) modem control. This shall have the effect of setting (not setting) CLOCAL in the **termios** *c_cflag* field, as defined in XBD Chapter 11 (on page 199).

It is unspecified whether *stty* shall report an error if an attempt to set a Control Mode fails.

Input Modes

ignbrk (**–ignbrk**) Ignore (do not ignore) break on input. This shall have the effect of setting (not setting) IGNBRK in the **termios** *c_iflag* field, as defined in XBD Chapter 11 (on page 199).

brkint (**–brkint**) Signal (do not signal) INTR on break. This shall have the effect of setting (not setting) BRKINT in the **termios** *c_iflag* field, as defined in XBD Chapter 11 (on page 199).

ignpar (**–ignpar**) Ignore (do not ignore) bytes with parity errors. This shall have the effect of setting (not setting) IGNPAR in the **termios** *c_iflag* field, as defined in XBD Chapter 11 (on page 199).

parmrk (**–parmrk**)
Mark (do not mark) parity errors. This shall have the effect of setting (not setting) PARMRK in the **termios** *c_iflag* field, as defined in XBD Chapter 11 (on page 199).

inpck (**–inpck**) Enable (disable) input parity checking. This shall have the effect of setting (not setting) INPCK in the **termios** *c_iflag* field, as defined in XBD Chapter 11 (on page 199).

istrip (**–istrip**) Strip (do not strip) input characters to seven bits. This shall have the effect of setting (not setting) ISTRIP in the **termios** *c_iflag* field, as defined in XBD Chapter 11 (on page 199).

inlcr (**–inlcr**) Map (do not map) NL to CR on input. This shall have the effect of setting (not setting) INLCR in the **termios** *c_iflag* field, as defined in XBD Chapter 11 (on page 199).

igncr (–igncr) Ignore (do not ignore) CR on input. This shall have the effect of setting (not setting) IGNCR in the **termios** *c_iflag* field, as defined in XBD Chapter 11 (on page 199).

icrnl (**–icrnl**) Map (do not map) CR to NL on input. This shall have the effect of setting (not setting) ICRNL in the **termios** *c_iflag* field, as defined in XBD Chapter 11 (on page 199).

ixon (**–ixon**) Enable (disable) START/STOP output control. Output from the system is stopped when the system receives STOP and started when the system receives START. This shall have the effect of setting (not setting) IXON in the **termios** *c_iflag* field, as defined in XBD Chapter 11 (on page 199).

ixany (**–ixany**) Allow any character to restart output. This shall have the effect of setting (not setting) IXANY in the **termios** *c_iflag* field, as defined in XBD Chapter 11 (on page 199).

ixoff (**–ixoff**) Request that the system send (not send) STOP characters when the input queue is nearly full and START characters to resume data transmission. This shall have the effect of setting (not setting) IXOFF in the **termios** *c_iflag* field, as defined in XBD Chapter 11 (on page 199).

Output Modes

opost (**–opost**) Post-process output (do not post-process output; ignore all other output modes). This shall have the effect of setting (not setting) OPOST in the **termios** *c_oflag* field, as defined in XBD Chapter 11 (on page 199).

XSI **onlcr** (**–onlcr**) Map (do not map) NL to CR-NL on output. This shall have the effect of setting (not setting) ONLCR in the **termios** *c_oflag* field, as defined in XBD Chapter 11 (on page 199).

XSI **ocrnl** (**–ocrnl**) Map (do not map) CR to NL on output. This shall have the effect of setting (not setting) OCRNL in the **termios** *c_oflag* field, as defined in XBD Chapter 11 (on page 199).

XSI **onocr** (**–onocr**) Do not (do) output CR at column zero. This shall have the effect of setting (not setting) ONOCR in the **termios** *c_oflag* field, as defined in XBD Chapter 11 (on page 199).

XSI **onlret** (**–onlret**) The terminal newline key performs (does not perform) the CR function. This shall have the effect of setting (not setting) ONLRET in the **termios** *c_oflag* field, as defined in XBD Chapter 11 (on page 199).

XSI **ofill** (**–ofill**) Use fill characters (use timing) for delays. This shall have the effect of setting (not setting) OFILL in the **termios** *c_oflag* field, as defined in XBD Chapter 11 (on page 199).

116173 XSI **ofdel** (**–ofdel**) Fill characters are DELs (NULs). This shall have the effect of setting (not
116174 setting) OFDEL in the **termios** *c_oflag* field, as defined in XBD Chapter 11 (on
116175 page 199).

116176 XSI **cr0 cr1 cr2 cr3** Select the style of delay for CRs. This shall have the effect of setting CRDLY to
116177 CR0, CR1, CR2, or CR3, respectively, in the **termios** *c_oflag* field, as defined in
116178 XBD Chapter 11 (on page 199).

116179 XSI **nl0 nl1** Select the style of delay for NL. This shall have the effect of setting NLDLY to
116180 NL0 or NL1, respectively, in the **termios** *c_oflag* field, as defined in XBD
116181 Chapter 11 (on page 199).

116182 XSI **tab0 tab1 tab2 tab3**
116183 Select the style of delay for horizontal tabs. This shall have the effect of setting
116184 TABDLY to TAB0, TAB1, TAB2, or TAB3, respectively, in the **termios** *c_oflag*
116185 field, as defined in XBD Chapter 11 (on page 199). Note that TAB3 has the
116186 effect of expanding <tab> characters to <space> characters.

116187 XSI **tabs** (**–tabs**) Synonym for **tab0** (**tab3**).

116188 XSI **bs0 bs1** Select the style of delay for <backspace> characters. This shall have the effect
116189 of setting BSDLY to BS0 or BS1, respectively, in the **termios** *c_oflag* field, as
116190 defined in XBD Chapter 11 (on page 199).

116191 XSI **ff0 ff1** Select the style of delay for <form-feed> characters. This shall have the effect
116192 of setting FFDLY to FF0 or FF1, respectively, in the **termios** *c_oflag* field, as
116193 defined in XBD Chapter 11 (on page 199).

116194 XSI **vt0 vt1** Select the style of delay for <vertical-tab> characters. This shall have the effect
116195 of setting VTDLY to VT0 or VT1, respectively, in the **termios** *c_oflag* field, as
116196 defined in XBD Chapter 11 (on page 199).

116197 **Local Modes**

116198 **isig** (**–isig**) Enable (disable) the checking of characters against the special control
116199 characters INTR, QUIT, and SUSP. This shall have the effect of setting (not
116200 setting) ISIG in the **termios** *c_lflag* field, as defined in XBD Chapter 11 (on
116201 page 199).

116202 **icanon** (**–icanon**) Enable (disable) canonical input (ERASE and KILL processing). This shall
116203 have the effect of setting (not setting) ICANON in the **termios** *c_lflag* field, as
116204 defined in XBD Chapter 11 (on page 199).

116205 **iexten** (**–iexten**) Enable (disable) any implementation-defined special control characters not
116206 currently controlled by **icanon**, **isig**, **ixon**, or **ixoff**. This shall have the effect of
116207 setting (not setting) IEXTEN in the **termios** *c_lflag* field, as defined in XBD
116208 Chapter 11 (on page 199).

116209 **echo** (**–echo**) Echo back (do not echo back) every character typed. This shall have the effect
116210 of setting (not setting) ECHO in the **termios** *c_lflag* field, as defined in XBD
116211 Chapter 11 (on page 199).

116212 **echoe** (**–echoe**) The ERASE character visually erases (does not erase) the last character in the
116213 current line from the display, if possible. This shall have the effect of setting
116214 (not setting) ECHOE in the **termios** *c_lflag* field, as defined in XBD Chapter 11
116215 (on page 199).

echok (–**echok**) Echo (do not echo) NL after KILL character. This shall have the effect of setting (not setting) ECHOK in the **termios** *c_lflag* field, as defined in XBD Chapter 11 (on page 199).

echonl (–**echonl**) Echo (do not echo) NL, even if **echo** is disabled. This shall have the effect of setting (not setting) ECHONL in the **termios** *c_lflag* field, as defined in XBD Chapter 11 (on page 199).

noflsh (–**noflsh**) Disable (enable) flush after INTR, QUIT, SUSP. This shall have the effect of setting (not setting) NOFLSH in the **termios** *c_lflag* field, as defined in XBD Chapter 11 (on page 199).

tostop (–**tostop**) Send SIGTTOU for background output. This shall have the effect of setting (not setting) TOSTOP in the **termios** *c_lflag* field, as defined in XBD Chapter 11 (on page 199).

Special Control Character Assignments

<*control*>-*character string*

Set <*control*>-*character* to *string*. If <*control*>-*character* is one of the character sequences in the first column of the following table, the corresponding XBD Chapter 11 (on page 199) control character from the second column shall be recognized. This has the effect of setting the corresponding element of the **termios** *c_cc* array (see XBD Chapter 14 (on page 221), **<termios.h>**).

Table 3-20 Control Character Names in *stty*

Control Character	c_cc Subscript	Description
eof	VEOF	EOF character
eol	VEOL	EOL character
erase	VERASE	ERASE character
intr	VINTR	INTR character
kill	VKILL	KILL character
quit	VQUIT	QUIT character
susp	VSUSP	SUSP character
start	VSTART	START character
stop	VSTOP	STOP character

If *string* is a single character, the control character shall be set to that character. If *string* is the two-character sequence `"^-"` or the string *undef*, the control character shall be set to _POSIX_VDISABLE , if it is in effect for the device; if _POSIX_VDISABLE is not in effect for the device, it shall be treated as an error. In the POSIX locale, if *string* is a two-character sequence beginning with <circumflex> (`'^'`), and the second character is one of those listed in the `"^c"` column of the following table, the control character shall be set to the corresponding character value in the Value column of the table.

116253 **Table 3-21** Circumflex Control Characters in *stty*

116254

^c	Value	^c	Value	^c	Value
a, A	<SOH>	l, L	<FF>	w, W	<ETB>
b, B	<STX>	m, M	<CR>	x, X	<CAN>
c, C	<ETX>	n, N	<SO>	y, Y	<EM>
d, D	<EOT>	o, O	<SI>	z, Z	<SUB>
e, E	<ENQ>	p, P	<DLE>	[	<ESC>
f, F	<ACK>	q, Q	<DC1>	\	<FS>
g, G	<BEL>	r, R	<DC2>	]	<GS>
h, H	<BS>	s, S	<DC3>	^	<RS>
i, I	<HT>	t, T	<DC4>	_	<US>
j, J	<LF>	u, U	<NAK>	?	<DEL>
k, K	<VT>	v, V	<SYN>		

116266 **min** *number*
116267 Set the value of MIN to *number*. MIN is used in non-canonical mode input processing
116268 (–**icanon**).

116269 **time** *number*
116270 Set the value of TIME to *number*. TIME is used in non-canonical mode input processing
116271 (–**icanon**).

116272 **Combination Modes**

116273 *saved settings*
116274 Set the current terminal characteristics to the saved settings produced by the –**g** option.

116275 **evenp** or **parity**
116276 Enable **parenb** and **cs7**; disable **parodd**.

116277 **oddp**
116278 Enable **parenb**, **cs7**, and **parodd**.

116279 –**parity**, –**evenp**, or –**oddp**
116280 Disable **parenb**, and set **cs8**.

116281 XSI **raw** (–**raw** or **cooked**)
116282 Enable (disable) raw input and output. Raw mode shall be equivalent to setting:

116283
```
stty cs8 erase ^- kill ^- intr ^- \
116284     quit ^- eof ^- eol ^- -post -inpck
```

116285 **nl** (–**nl**)
116286 Disable (enable) **icrnl**. In addition, –**nl** unsets **inlcr** and **igncr**.

116287 **ek** Reset ERASE and KILL characters back to system defaults.

116288 **sane**
116289 Reset all modes to some reasonable, unspecified, values.

Terminal Window Size

rows *number*
Set the number of rows in the terminal window size to the number given.

cols *number*
Set the number of columns in the terminal window size to the number given.

The terminal window size shall be updated as if the *stty* utility calls *tcgetwinsize*( ) to populate a **winsize** structure, updates one or both of the *ws_row* and *ws_col* members according to the **rows** and **cols** numbers specified, and then calls *tcsetwinsize*( ) with the updated structure (see XSH *tcgetwinsize*( ) and *tcsetwinsize*( )).

Informational Queries

size
Write the current terminal window size to standard output.

STDIN

Although no input is read from standard input, standard input shall be used to get the current terminal I/O characteristics and to set new terminal I/O characteristics.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *stty*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE This variable determines the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments) and which characters are in the class **print**.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If operands are specified and they do not include any Informational Queries, no output shall be produced.

If the **size** operand is specified, *stty* shall write to standard output the terminal window size as follows:

`"%1dΔ%1d\n", <rows>, <columns>`

where <*rows*> and <*columns*> are the number of rows and columns in the terminal window size, respectively.

If the **–g** option is specified, *stty* shall write to standard output the current settings in a form that

116332 can be used as arguments to another instance of *stty* on the same system.

116333 If the −**a** option is specified, all of the information as described in the OPERANDS section shall
116334 be written to standard output. Unless otherwise specified, this information shall be written as
116335 <space>-separated tokens in an unspecified format, on one or more lines, with an unspecified
116336 number of tokens per line. Additional information may be written.

116337 If no options or operands are specified, an unspecified subset of the information written for the
116338 −**a** option shall be written.

116339 If speed information is written as part of the default output, or if the −**a** option is specified and if
116340 the terminal input speed and output speed are the same, the speed information shall be written
116341 as follows:

116342 `"speed %d baud;", <speed>`

116343 Otherwise, speeds shall be written as:

116344 `"ispeed %d baud; ospeed %d baud;", <ispeed>, <ospeed>`

116345 In locales other than the POSIX locale, the word **baud** may be changed to something more
116346 appropriate in those locales.

116347 If control characters are written as part of the default output, or if the −**a** option is specified,
116348 control characters shall be written as:

116349 `"%s = %s;", <control-character name>, <value>`

116350 where <*value*> is either the character, or some visual representation of the character if it is non-
116351 printable, or the string `"<undef>"` if the character is disabled.

116352 **STDERR**

116353 The standard error shall be used only for diagnostic messages.

116354 **OUTPUT FILES**

116355 None.

116356 **EXTENDED DESCRIPTION**

116357 None.

116358 **EXIT STATUS**

116359 The following exit values shall be returned:

116360 0 Successful completion.

116361 >0 An error occurred.

116362 **CONSEQUENCES OF ERRORS**

116363 Default.

116364 **APPLICATION USAGE**

116365 The −**g** flag is designed to facilitate the saving and restoring of terminal state from the shell level.
116366 For example, a program may:

```
116367 saveterm=$(stty -g)                        # save terminal state
116368 restoresize=$(
116369     printf "stty rows %d cols %d" $(stty size)
116370 )                                          # save terminal size
116371 stty new settings                          # set new state
116372 ...
116373 [ -n "$saveterm" ] && stty "$saveterm"    # restore terminal state
```



```
eval "$restoresize"                          # restore terminal size
```

Since the format is unspecified, the saved value is not portable across systems.

Since the –**a** format is so loosely specified, scripts that save and restore terminal settings should use the –**g** option.

EXAMPLES

None.

RATIONALE

The original *stty* description was taken directly from System V and reflected the System V terminal driver **termio**. It has been modified to correspond to the terminal driver **termios**.

Output modes are specified only for XSI-conformant systems. All implementations are expected to provide *stty* operands corresponding to all of the output modes they support.

The *stty* utility is primarily used to tailor the user interface of the terminal, such as selecting the preferred ERASE and KILL characters. As an application programming utility, *stty* can be used within shell scripts to alter the terminal settings for the duration of the script.

The **termios** section states that individual disabling of control characters is possible through the option _POSIX_VDISABLE. If enabled, two conventions currently exist for specifying this: System V uses `"^-"`, and BSD uses *undef*. Both are accepted by *stty* in this volume of POSIX.1-2024. The other BSD convention of using the letter `'u'` was rejected because it conflicts with the actual letter `'u'`, which is an acceptable value for a control character.

Early proposals did not specify the mapping of `"^c"` to control characters because the control characters were not specified in the POSIX locale character set description file requirements. The control character set is now specified in XBD Chapter 3 (on page 31), so the historical mapping is specified. Note that although the mapping corresponds to control-character key assignments on many terminals that use the ISO/IEC 646:1991 standard (or ASCII) character encodings, the mapping specified here is to the control characters, not their keyboard encodings.

Since **termios** supports separate speeds for input and output, two new options were added to specify each distinctly.

Some historical implementations use standard input to get and set terminal characteristics; others use standard output. Since input from a login TTY is usually restricted to the owner while output to a TTY is frequently open to anyone, using standard input provides fewer chances of accidentally (or maliciously) altering the terminal settings of other users. Using standard input also allows *stty* –**a** and *stty* –**g** output to be redirected for later use. Therefore, usage of standard input is required by this volume of POSIX.1-2024.

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 2 (on page 2472)

XBD Chapter 8 (on page 167), Chapter 11 (on page 199), Section 12.2 (on page 215), **<termios.h>**

CHANGE HISTORY

First released in Issue 2.

Issue 5

The description of **tabs** is clarified.

The FUTURE DIRECTIONS section is added.

Issue 6

The LEGACY items **iuclc** (–**iuclc**), **xcase** (–**xcase**), **olcuc** (–**olcuc**), **lcase** (–**lcase**), and **LCASE** (–**LCASE**) are removed.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/37 is applied, applying IEEE PASC Interpretation 1003.2 #133, fixing an error in the OPERANDS section for the Combination Modes **nl** (–**nl**).

Issue 7

Austin Group Interpretation 1003.1-2001 #144 is applied, moving functionality relating to the IXANY symbol from the XSI option to the Base.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0189 [908] is applied.

Issue 8

Austin Group Defects 1053, 1532, and 1687 are applied, changing the –**g** option and adding the **rows** *number*, **cols** *number*, and **size** operands.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1508 is applied, changing the EXIT STATUS section.

Austin Group Defect 1604 is applied, changing *undef* to `"<undef>"` in the STDOUT section, and changing **icanon** to –**icanon** in the descriptions of **min** and **time**.

NAME

tabs — set terminal tabs

SYNOPSIS

XSI `tabs [-n|-a|-a2|-c|-c2|-c3|-f|-p|-s|-u] [-T type]`

`tabs [-T type] n[[sep[+]n]...]`

DESCRIPTION

The *tabs* utility shall display a series of characters that first clears the hardware terminal tab settings and then initializes the tab stops at the specified positions and optionally adjusts the margin.

The phrase ``tab-stop position *N*'' shall be taken to mean that, from the start of a line of output, tabbing to position *N* shall cause the next character output to be in the (*N*+1)th column position on that line. The maximum number of tab stops allowed is terminal-dependent.

It need not be possible to implement *tabs* on certain terminals. If the terminal type obtained from the *TERM* environment variable or **–T** option represents such a terminal, an appropriate diagnostic message shall be written to standard error and *tabs* shall exit with a status greater than zero.

OPTIONS

The *tabs* utility shall conform to XBD Section 12.2 (on page 215), except for various extensions: the options **–a2**, **–c2**, and **–c3** are multi-character.

The following options shall be supported:

–n Specify repetitive tab stops separated by a uniform number of column positions, *n*, where *n* is a single-digit decimal number. The default usage of *tabs* with no arguments shall be equivalent to *tabs* –8. When **–0** is used, the tab stops shall be cleared and no new ones set.

XSI **–a** 1,10,16,36,72
Assembler, applicable to some mainframes.

XSI **–a2** 1,10,16,40,72
Assembler, applicable to some mainframes.

XSI **–c** 1,8,12,16,20,55
COBOL, normal format.

XSI **–c2** 1,6,10,14,49
COBOL, compact format (columns 1 to 6 omitted).

XSI **–c3** 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1 to 6 omitted), with more tabs than **–c2**.

XSI **–f** 1,7,11,15,19,23
FORTRAN

XSI **–p** 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
PL/1

XSI **–s** 1,10,55
SNOBOL

XSI **–u** 1,12,20,44
Assembler, applicable to some mainframes.

116477 **–T** *type* Indicate the type of terminal. If this option is not supplied and the *TERM* variable
116478 is unset or null, an unspecified default terminal type shall be used. The setting of
116479 *type* shall take precedence over the value in *TERM*.

116480 **OPERANDS**

116481 The following operand shall be supported:

116482 *n*[[*sep*[+]*n*]...] A single command line argument that consists of one or more tab-stop values (*n*)
116483 separated by a separator character (*sep*) which is either a <comma> or a <blank>
116484 character. The application shall ensure that the tab-stop values are positive decimal
116485 integers in strictly ascending order. If any tab-stop value (except the first one) is
116486 preceded by a <plus-sign>, it is taken as an increment to be added to the previous
116487 value. For example, the tab lists 1,10,20,30 and `"1 10 +10 +10"` are considered
116488 to be identical.

116489 **STDIN**

116490 Not used.

116491 **INPUT FILES**

116492 None.

116493 **ENVIRONMENT VARIABLES**

116494 The following environment variables shall affect the execution of *tabs*:

116495 *LANG* Provide a default value for the internationalization variables that are unset or null.
116496 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
116497 variables used to determine the values of locale categories.)

116498 *LC_ALL* If set to a non-empty string value, override the values of all the other
116499 internationalization variables.

116500 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
116501 characters (for example, single-byte as opposed to multi-byte characters in
116502 arguments).

116503 *LC_MESSAGES*
116504 Determine the locale that should be used to affect the format and contents of
116505 diagnostic messages written to standard error.

116506 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

116507 *TERM* Determine the terminal type. If this variable is unset or null, and if the **–T** option is
116508 not specified, an unspecified default terminal type shall be used.

116509 **ASYNCHRONOUS EVENTS**

116510 Default.

116511 **STDOUT**

116512 If standard output is a terminal, the appropriate sequence to clear and set the tab stops may be
116513 written to standard output in an unspecified format. If standard output is not a terminal,
116514 undefined results occur.

116515 **STDERR**

116516 The standard error shall be used only for diagnostic messages.

116517 **OUTPUT FILES**

116518 None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

This utility makes use of the terminal's hardware tabs and the *stty tabs* option.

This utility is not recommended for application use.

Some integrated display units might not have escape sequences to set tab stops, but may be set by internal system calls. On these terminals, *tabs* works if standard output is directed to the terminal; if output is directed to another file, however, *tabs* fails.

EXAMPLES

None.

RATIONALE

Consideration was given to having the *tput* utility handle all of the functions described in *tabs*. However, the separate *tabs* utility was retained because it seems more intuitive to use a command named *tabs* than *tput* with a new option. The *tput* utility does not support setting or clearing tabs, and no known historical version of *tabs* supports the capability of setting arbitrary tab stops.

The System V *tabs* interface is very complex; the version in this volume of POSIX.1-2024 has a reduced feature list, but many of the features omitted were restored as part of the XSI option even though the supported languages and coding styles are primarily historical.

There was considerable sentiment for specifying only a means of resetting the tabs back to a known state—presumably the ``standard'' of tabs every eight positions. The following features were omitted:

- Setting tab stops via the first line in a file, using *−−file*. Since even the SVID has no complete explanation of this feature, it is doubtful that it is in widespread use.

In an early proposal, a **−t** *tablist* option was added for consistency with *expand*; this was later removed when inconsistencies with the historical list of tabs were identified.

Consideration was given to adding a **−p** option that would output the current tab settings so that they could be saved and then later restored. This was not accepted because querying the tab stops of the terminal is not a capability in historical *terminfo* or *termcap* facilities and might not be supported on a wide range of terminals.

FUTURE DIRECTIONS

None.

SEE ALSO

expand, *stty*, *tput*, *unexpand*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

The *tabs* utility is removed from the User Portability Utilities option. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The SYNOPSIS and OPERANDS sections are updated.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

tail — copy the last part of a file

SYNOPSIS

```
tail [-f] [-c number|-n number] [file]
tail -r [-n number] [file]
```

DESCRIPTION

The *tail* utility shall copy its input file to the standard output beginning at a designated place.

Copying shall begin at the point in the file indicated by the **−c** *number* or **−n** *number* options. The option-argument *number* shall be counted in units of lines or bytes, according to the options **−n** and **−c**. Both line and byte counts start from 1.

Tails relative to the end of the file may be saved in an internal buffer, and thus may be limited in length. Such a buffer, if any, shall be no smaller than {LINE_MAX}*10 bytes.

OPTIONS

The *tail* utility shall conform to XBD Section 12.2 (on page 215), except that `'+'` may be recognized as an option delimiter as well as `'−'`.

The following options shall be supported:

−c *number* The application shall ensure that the *number* option-argument is a decimal integer, optionally including a sign. The sign shall affect the location in the file, measured in bytes, to begin the copying:

Sign	Copying Starts
+	Relative to the beginning of the file.
−	Relative to the end of the file.
none	Relative to the end of the file.

The application shall ensure that if the sign of the *number* option-argument is `'+'`, the *number* option-argument is a non-zero decimal integer.

The origin for counting shall be 1; that is, **−c** +1 represents the first byte of the file, **−c** −1 the last.

−f If the input file is a regular file or if the *file* operand specifies a FIFO, do not terminate after the last line of the input file has been copied, but read and copy further bytes from the input file when they become available. If no *file* operand is specified and standard input is a pipe or FIFO, the **−f** option shall be ignored. If the input file is not a FIFO, pipe, or regular file, it is unspecified whether or not the **−f** option shall be ignored.

−n *number* If **−r** is not specified, this option shall be equivalent to **−c** *number*, except the starting location in the file shall be measured in lines instead of bytes. The origin for counting shall be 1; that is, **−n** +1 represents the first line of the file, **−n** −1 the last.

If **−r** is specified, *number* shall specify the number of lines to read (in reverse) from the end of the input file. The application shall ensure that *number* does not have a sign.

−r Copy the lines in reverse order (last line first). If **−n** is specified, that many lines of the file, starting with the last line, shall be copied. If **−n** is not specified, every line of the input file shall be copied.

If none of the **−c**, **−n** or **−r** options is specified, **−n** 10 shall be assumed.

OPERANDS

The following operand shall be supported:

file A pathname of an input file. If no *file* operand is specified, the standard input shall be used.

STDIN

The standard input shall be used if no *file* operand is specified, and shall be used if the *file* operand is `'−'` and the implementation treats the `'−'` as meaning standard input. Otherwise, the standard input shall not be used. See the INPUT FILES section.

INPUT FILES

If the **−c** option is specified, the input file can contain arbitrary data; otherwise, the input file shall be a text file.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *tail*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The designated portion of the input file shall be written to standard output.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The **−c** option should be used with caution when the input is a text file containing multi-byte characters; it may produce output that does not start on a character boundary.

Although the input file to *tail* can be any type, the results might not be what would be expected on some character special device files or on file types not described by the System Interfaces volume of POSIX.1-2024. Since this volume of POSIX.1-2024 does not specify the block size used when doing input, *tail* need not read all of the data from devices that only perform block transfers.

When using *tail* to process pathnames, and the **−c** option is not specified, it is recommended that LC_ALL, or at least LC_CTYPE and LC_COLLATE, are set to POSIX or C in the environment, since pathnames can contain byte sequences that do not form valid characters in some locales, in which case the utility's behavior would be undefined. In the POSIX locale each byte is a valid single-byte character, and therefore this problem is avoided.

EXAMPLES

The **−f** option can be used to monitor the growth of a file that is being written by some other process. For example, the command:

```
tail -f fred
```

prints the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed. As another example, the command:

```
tail -f -c 15 fred
```

prints the last 15 bytes of the file **fred**, followed by any bytes that are appended to **fred** between the time *tail* is initiated and killed.

RATIONALE

This version of *tail* was created to allow conformance to the Utility Syntax Guidelines. The historical **−b** option was omitted because of the general non-portability of block-sized units of text. The **−c** option historically meant ``characters'', but this volume of POSIX.1-2024 indicates that it means ``bytes''. This was selected to allow reasonable implementations when multi-byte characters are possible; it was not named **−b** to avoid confusion with the historical **−b**.

The origin of counting both lines and bytes is 1, matching all widespread historical implementations. Hence *tail* **−n** +0 is not conforming usage because it attempts to output line zero; but note that *tail* **−n** 0 does conform, and outputs nothing.

Earlier versions of this standard allowed the following forms in the SYNOPSIS:

```
tail -[number][b|c|l][f] [file]
tail +[number][b|c|l][f] [file]
```

These forms are no longer specified by POSIX.1-2024, but may be present in some implementations.

The restriction on the internal buffer is a compromise between the historical System V implementation of 4 096 bytes and the BSD 32 768 bytes.

The **−f** option has been implemented as a loop that sleeps for 1 second and copies any bytes that are available. This is sufficient, but if more efficient methods of determining when new data are available are developed, implementations are encouraged to use them.

Historical documentation indicates that *tail* ignores the **−f** option if the input file is a pipe (pipe

and FIFO on systems that support FIFOs). On BSD-based systems, this has been true; on System V-based systems, this was true when input was taken from standard input, but it did not ignore the **−f** flag if a FIFO was named as the *file* operand. Since the **−f** option is not useful on pipes and all historical implementations ignore **−f** if no *file* operand is specified and standard input is a pipe, this volume of POSIX.1-2024 requires this behavior. However, since the **−f** option is useful on a FIFO, this volume of POSIX.1-2024 also requires that if a FIFO is named, the **−f** option shall not be ignored. Earlier versions of this standard did not state any requirement for the case where no *file* operand is specified and standard input is a FIFO. The standard has been updated to reflect current practice which is to treat this case the same as a pipe on standard input. Although historical behavior does not ignore the **−f** option for other file types, this is unspecified so that implementations are allowed to ignore the **−f** option if it is known that the file cannot be extended.

The functionality made available by *tail* **−r** has been historically provided on some systems by a separate utility (*tac*), although *tac* traditionally lacked support for **−n** to limit the output. While both `tail -n$n | tac` and `tac | head -n$n` can be used to output a fixed length of reversed line output, the standard developers decided that it was preferable to have a single utility `tail -r -n$n` for the same purpose. Furthermore, in deciding whether to standardize *tac* rather than *tail* **−r**, it was determined that more implementations that have achieved POSIX certification had already implemented *tail* **−r** as an extension.

FUTURE DIRECTIONS

None.

SEE ALSO

head

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The obsolescent SYNOPSIS lines and associated text are removed.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that `'+'` may be recognized as an option delimiter in the OPTIONS section.

Austin Group Interpretation 1003.1-2001 #092 is applied.

Austin Group Interpretation 1003.1-2001 #100 is applied, adding the requirement on applications that if the sign of the option-argument *number* is `'+'`, the *number* option-argument is a non-zero decimal integer.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

SD5-XCU-ERN-114 is applied, updating the OPTIONS section (the **−f** option).

SD5-XCU-ERN-149 is applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0190 [663] is applied.

Issue 8

Austin Group Defect 877 is applied, adding the **−r** option.

116741 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

talk — talk to another user

SYNOPSIS

UP

```
talk address [terminal]
```

DESCRIPTION

The *talk* utility is a two-way, screen-oriented communication program.

When first invoked, *talk* shall send a message similar to:

```
Message from <unspecified string>
talk: connection requested by your_address
talk: respond with: talk your_address
```

to the specified *address*. At this point, the recipient of the message can reply by typing:

```
talk your_address
```

Once communication is established, the two parties can type simultaneously, with their output displayed in separate regions of the screen. Characters shall be processed as follows:

- Typing the <alert> character shall alert the recipient's terminal.
- Typing <control>-L shall cause the sender's screen regions to be refreshed.
- Typing the erase and kill characters shall affect the sender's terminal in the manner described by the **termios** interface in XBD Chapter 11 (on page 199).
- Typing the interrupt or end-of-file characters shall terminate the local *talk* utility. Once the *talk* session has been terminated on one side, the other side of the *talk* session shall be notified that the *talk* session has been terminated and shall be able to do nothing except exit.
- Typing characters from *LC_CTYPE* classifications **print** or **space** shall cause those characters to be sent to the recipient's terminal.
- When and only when the *stty* **iexten** local mode is enabled, the existence and processing of additional special control characters and multi-byte or single-byte functions shall be implementation-defined.
- Typing other non-printable characters shall cause implementation-defined sequences of printable characters to be sent to the recipient's terminal.

Permission to be a recipient of a *talk* message can be denied or granted by use of the *mesg* utility. However, a user's privilege may further constrain the domain of accessibility of other users' terminals. The *talk* utility shall fail when the user lacks appropriate privileges to perform the requested action.

Certain block-mode terminals do not have all the capabilities necessary to support the simultaneous exchange of messages required for *talk*. When this type of exchange cannot be supported on such terminals, the implementation may support an exchange with reduced levels of simultaneous interaction or it may report an error describing the terminal-related deficiency.

OPTIONS

None.

116781 **OPERANDS**

116782 The following operands shall be supported:

116783 *address* The recipient of the *talk* session. One form of *address* is the <*user name*>, as returned
116784 by the *who* utility. Other address formats and how they are handled are
116785 unspecified.

116786 *terminal* If the recipient is logged in more than once, the *terminal* argument can be used to
116787 indicate the appropriate terminal name. If *terminal* is not specified, the *talk* message
116788 shall be displayed on one or more accessible terminals in use by the recipient. The
116789 format of *terminal* shall be the same as that returned by the *who* utility.

116790 **STDIN**

116791 Characters read from standard input shall be copied to the recipient's terminal in an unspecified
116792 manner. If standard input is not a terminal, talk shall write a diagnostic message and exit with a
116793 non-zero status.

116794 **INPUT FILES**

116795 None.

116796 **ENVIRONMENT VARIABLES**

116797 The following environment variables shall affect the execution of *talk*:

116798 *LANG* Provide a default value for the internationalization variables that are unset or null.
116799 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
116800 variables used to determine the values of locale categories.)

116801 *LC_ALL* If set to a non-empty string value, override the values of all the other
116802 internationalization variables.

116803 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
116804 characters (for example, single-byte as opposed to multi-byte characters in
116805 arguments and input files). If the recipient's locale does not use an *LC_CTYPE*
116806 equivalent to the sender's, the results are undefined.

116807 *LC_MESSAGES*

116808 Determine the locale that should be used to affect the format and contents of
116809 diagnostic messages written to standard error and informative messages written to
116810 standard output.

116811 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

116812 *TERM* Determine the name of the invoker's terminal type. If this variable is unset or null,
116813 an unspecified default terminal type shall be used.

116814 **ASYNCHRONOUS EVENTS**

116815 When the *talk* utility receives a SIGINT signal, the utility shall terminate and exit with a zero
116816 status. It shall take the standard action for all other signals.

116817 **STDOUT**

116818 If standard output is a terminal, characters copied from the recipient's standard input may be
116819 written to standard output. Standard output also may be used for diagnostic messages. If
116820 standard output is not a terminal, *talk* shall exit with a non-zero status.

116821 **STDERR**

116822 None.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred or *talk* was invoked on a terminal incapable of supporting it.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Because the handling of non-printable, non-<space> characters is tied to the *stty* description of **iexten**, implementation extensions within the terminal driver can be accessed. For example, some implementations provide line editing functions with certain control character sequences.

EXAMPLES

None.

RATIONALE

The *write* utility was included in this volume of POSIX.1-2024 since it can be implemented on all terminal types. The *talk* utility, which cannot be implemented on certain terminals, was considered to be a ``better'' communications interface. Both of these programs are in widespread use on historical implementations. Therefore, both utilities have been specified.

All references to networking abilities (*talk*ing to a user on another system) were removed as being outside the scope of this volume of POSIX.1-2024.

Historical BSD and System V versions of *talk* terminate both of the conversations when either user breaks out of the session. This can lead to adverse consequences if a user unwittingly continues to enter text that is interpreted by the shell when the other terminates the session. Therefore, the version of *talk* specified by this volume of POSIX.1-2024 requires both users to terminate their end of the session explicitly.

Only messages sent to the terminal of the invoking user can be internationalized in any way:

- The original ``Message from <*unspecified string*> ...'' message sent to the terminal of the recipient cannot be internationalized because the environment of the recipient is as yet inaccessible to the *talk* utility. The environment of the invoking party is irrelevant.
- Subsequent communication between the two parties cannot be internationalized because the two parties may specify different languages in their environment (and non-portable characters cannot be mapped from one language to another).
- Neither party can be required to communicate in a language other than C and/or the one specified by their environment because unavailable terminal hardware support (for example, fonts) may be required.

The text in the STDOUT section reflects the usage of the verb ``display'' in this section; some *talk* implementations actually use standard output to write to the terminal, but this volume of POSIX.1-2024 does not require that to be the case.

The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write* require that they all use or accept the same format.

The handling of non-printable characters is partially implementation-defined because the details of mapping them to printable sequences is not needed by the user. Historical implementations, for security reasons, disallow the transmission of non-printable characters that may send commands to the other terminal.

FUTURE DIRECTIONS

None.

SEE ALSO

mesg, *stty*, *who*, *write*

XBD Chapter 8 (on page 167), Chapter 11 (on page 199)

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is marked as part of the User Portability Utilities option.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

tee — duplicate standard input

SYNOPSIS

```
tee [-ai] [file...]
```

DESCRIPTION

The *tee* utility shall copy standard input to standard output, making a copy in zero or more files. The *tee* utility shall not buffer output.

If the **–a** option is not specified, output files shall be written (see Section 1.1.1.4, on page 2454).

OPTIONS

The *tee* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–a Append the output to the files.

–i Ignore the SIGINT signal.

OPERANDS

The following operands shall be supported:

file A pathname of an output file. If a *file* operand is '–', it shall refer to a file named –; implementations shall not treat it as meaning standard output. Processing of at least 13 *file* operands shall be supported.

STDIN

The standard input can be of any type.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *tee*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default, except that if the **–i** option was specified, SIGINT shall be ignored.

STDOUT

The standard output shall be a copy of the standard input.

116921 **STDERR**

116922 The standard error shall be used only for diagnostic messages.

116923 **OUTPUT FILES**

116924 If any *file* operands are specified, the standard input shall be copied to each named file.

116925 **EXTENDED DESCRIPTION**

116926 None.

116927 **EXIT STATUS**

116928 The following exit values shall be returned:

116929 0 The standard input was successfully copied to all output files.

116930 >0 An error occurred.

116931 **CONSEQUENCES OF ERRORS**

116932 If a write to any successfully opened *file* operand fails, writes to other successfully opened *file*
116933 operands and standard output shall continue, but the exit status shall be non-zero. Otherwise,
116934 the default actions specified in Section 1.4 (on page 2462) apply.

116935 **APPLICATION USAGE**

116936 The *tee* utility is usually used in a pipeline, to make a copy of the output of some utility.

116937 The *file* operand is technically optional, but *tee* is no more useful than *cat* when none is specified.

116938 **EXAMPLES**

116939 Save an unsorted intermediate form of the data in a pipeline:

116940 ```
... | tee unsorted | sort > sorted
```

116941 **RATIONALE**

116942 The buffering requirement means that *tee* is not allowed to use ISO C standard fully buffered or
116943 line-buffered writes. It does not mean that *tee* has to do 1-byte reads followed by 1-byte writes.

116944 It should be noted that early versions of BSD ignore any invalid options and accept a single `'-'`
116945 as an alternative to **–i**. They also print a message if unable to open a file:

116946 ```
"tee: cannot access %s\n", <pathname>
```

116947 Historical implementations ignore write errors. This is explicitly not permitted by this volume of
116948 POSIX.1-2024.

116949 Some historical implementations use O_APPEND when providing append mode; others use the
116950 *lseek*( ) function to seek to the end-of-file after opening the file without O_APPEND. This volume
116951 of POSIX.1-2024 requires functionality equivalent to using O_APPEND; see Section 1.1.1.4 (on
116952 page 2454).

116953 **FUTURE DIRECTIONS**

116954 If this utility is directed to create a new directory entry that contains any bytes that have the
116955 encoded value of a <newline> character, implementations are encouraged to treat this as an
116956 error. A future version of this standard may require implementations to treat this as an error.

116957 **SEE ALSO**

116958 Chapter 1 (on page 2453), *cat*

116959 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

116960 XSH *lseek*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

IEEE PASC Interpretation 1003.2 #168 is applied.

Issue 7

Austin Group Interpretation 1003.1-2001 #092 is applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1494 is applied, inserting a missing closing parenthesis.

NAME

test — evaluate expression

SYNOPSIS

```
test [expression]
[ [expression] ]
```

DESCRIPTION

The *test* utility shall evaluate the *expression* and indicate the result of the evaluation by its exit status. An exit status of zero indicates that the expression evaluated as true and an exit status of 1 indicates that the expression evaluated as false.

In the second form of the utility, where the utility name used is *[* rather than *test*, the application shall ensure that the closing square bracket is a separate argument. The *test* and *[* utilities may be implemented as a single linked utility which examines the basename of the zeroth command line argument to determine whether to behave as the *test* or *[* variant. Applications using the *exec* family of functions to execute these utilities shall ensure that the argument passed in *arg0* or *argv*[0] is `'['` when executing the *[* utility and has a basename of `"test"` when executing the *test* utility.

OPTIONS

The *test* utility shall not recognize the `"--"` argument in the manner specified by Guideline 10 in XBD Section 12.2 (on page 215). In addition, when the utility name used is *[* the utility does not conform to Guidelines 1 and 2.

No options shall be supported.

OPERANDS

The application shall ensure that all operators and elements of primaries are presented as separate arguments to the *test* utility.

The following primaries can be used to construct *expression*:

−b *pathname* True if *pathname* resolves to an existing directory entry for a block special file. False if *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry for a file that is not a block special file.

−c *pathname* True if *pathname* resolves to an existing directory entry for a character special file. False if *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry for a file that is not a character special file.

−d *pathname* True if *pathname* resolves to an existing directory entry for a directory. False if *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry for a file that is not a directory.

−e *pathname* True if *pathname* resolves to an existing directory entry. False if *pathname* cannot be resolved.

pathname1 **−ef** *pathname2*
True if *pathname1* and *pathname2* resolve to existing directory entries for the same file; otherwise, false.

−f *pathname* True if *pathname* resolves to an existing directory entry for a regular file. False if *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry for a file that is not a regular file.

−g *pathname* True if *pathname* resolves to an existing directory entry for a file that has its set-group-ID flag set. False if *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry for a file that does not have its set-group-ID flag set.

117018 –**h** *pathname* True if *pathname* resolves to an existing directory entry for a symbolic link. False if
117019 *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry
117020 for a file that is not a symbolic link. If the final component of *pathname* is a
117021 symbolic link, that symbolic link is not followed.

117022 –**L** *pathname* True if *pathname* resolves to an existing directory entry for a symbolic link. False if
117023 *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry
117024 for a file that is not a symbolic link. If the final component of *pathname* is a
117025 symbolic link, that symbolic link is not followed.

117026 –**n** *string* True if the length of *string* is non-zero; otherwise, false.

117027 *pathname1* –**nt** *pathname2*
117028 True if *pathname1* resolves to an existing file and *pathname2* cannot be resolved, or if
117029 both resolve to existing files and *pathname1* is newer than *pathname2* according to
117030 their last data modification timestamps; otherwise, false.

117031 *pathname1* –**ot** *pathname2*
117032 True if *pathname2* resolves to an existing file and *pathname1* cannot be resolved, or if
117033 both resolve to existing files and *pathname1* is older than *pathname2* according to
117034 their last data modification timestamps; otherwise, false.

117035 –**p** *pathname* True if *pathname* resolves to an existing directory entry for a FIFO. False if *pathname*
117036 cannot be resolved, or if *pathname* resolves to an existing directory entry for a file
117037 that is not a FIFO.

117038 –**r** *pathname* True if *pathname* resolves to an existing directory entry for a file for which
117039 permission to read from the file is granted, as defined in Section 1.1.1.4 (on page
117040 2454). False if *pathname* cannot be resolved, or if *pathname* resolves to an existing
117041 directory entry for a file for which permission to read from the file is not granted.

117042 –**S** *pathname* True if *pathname* resolves to an existing directory entry for a socket. False if
117043 *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry
117044 for a file that is not a socket.

117045 –**s** *pathname* True if *pathname* resolves to an existing directory entry for a file that has a size
117046 greater than zero. False if *pathname* cannot be resolved, or if *pathname* resolves to an
117047 existing directory entry for a file that does not have a size greater than zero.

117048 –**t** *file_descriptor*
117049 True if file descriptor number *file_descriptor* is open and is associated with a
117050 terminal. False if *file_descriptor* is not a valid file descriptor number, or if file
117051 descriptor number *file_descriptor* is not open, or if it is open but is not associated
117052 with a terminal.

117053 –**u** *pathname* True if *pathname* resolves to an existing directory entry for a file that has its set-
117054 user-ID flag set. False if *pathname* cannot be resolved, or if *pathname* resolves to an
117055 existing directory entry for a file that does not have its set-user-ID flag set.

117056 –**w** *pathname* True if *pathname* resolves to an existing directory entry for a file for which
117057 permission to write to the file is granted, as defined in Section 1.1.1.4 (on page
117058 2454). False if *pathname* cannot be resolved, or if *pathname* resolves to an existing
117059 directory entry for a file for which permission to write to the file is not granted.

117060 –**x** *pathname* True if *pathname* resolves to an existing directory entry for a file for which
117061 permission to execute the file (or search it, if it is a directory) is granted, as defined
117062 in Section 1.1.1.4 (on page 2454). False if *pathname* cannot be resolved, or if
117063 *pathname* resolves to an existing directory entry for a file for which permission to

117064 execute (or search) the file is not granted.

117065 **–z** *string* True if the length of string *string* is zero; otherwise, false.

117066 *string* True if the string *string* is not the null string; otherwise, false.

117067 *s1* = *s2* True if the strings *s1* and *s2* are identical; otherwise, false.

117068 *s1* **!=** *s2* True if the strings *s1* and *s2* are not identical; otherwise, false.

117069 *s1* **>** *s2* True if *s1* collates after *s2* in the current locale; otherwise, false.

117070 *s1* **<** *s2* True if *s1* collates before *s2* in the current locale; otherwise, false.

117071 *n1* **–eq** *n2* True if the integers *n1* and *n2* are algebraically equal; otherwise, false.

117072 *n1* **–ne** *n2* True if the integers *n1* and *n2* are not algebraically equal; otherwise, false.

117073 *n1* **–gt** *n2* True if the integer *n1* is algebraically greater than the integer *n2*; otherwise, false.

117074 *n1* **–ge** *n2* True if the integer *n1* is algebraically greater than or equal to the integer *n2*;
117075 otherwise, false.

117076 *n1* **–lt** *n2* True if the integer *n1* is algebraically less than the integer *n2*; otherwise, false.

117077 *n1* **–le** *n2* True if the integer *n1* is algebraically less than or equal to the integer *n2*; otherwise,
117078 false.

117079 With the exception of the **–h** *pathname* and **–L** *pathname* primaries, if a *pathname*, *pathname1*, or
117080 *pathname2* argument is a symbolic link, *test* shall evaluate the expression by resolving the
117081 symbolic link and using the file referenced by the link.

117082 These primaries can be combined with the following operator:

117083 **!** *expression* True if *expression* is false. False if *expression* is true.

117084 The primaries with two elements of the form:

117085 `-primary_operator primary_operand`

117086 are known as *unary primaries*. The primaries with three elements in either of the two forms:

117087 `primary_operand -primary_operator primary_operand`

117088 `primary_operand primary_operator primary_operand`

117089 are known as *binary primaries*. Additional implementation-defined operators and
117090 *primary_operator*s may be provided by implementations. They shall be of the form *–operator*
117091 where the first character of *operator* is not a digit.

117092 The algorithm for determining the precedence of the operators and the return value that shall be
117093 generated is based on the number of arguments presented to *test*. (However, when using the
117094 `"[...]"` form, the <right-square-bracket> final argument shall not be counted in this
117095 algorithm.)

117096 In the following list, $1, $2, $3, and $4 represent the arguments presented to *test*:

117097 0 arguments: Exit false (1).

117098 1 argument: Exit true (0) if $1 is not null; otherwise, exit false.

117099 2 arguments:

- If $1 is `'!'`, exit true if $2 is null, false if $2 is not null.

- If $1 is a unary primary, exit true if the unary test is true, false if the unary test is false.
- Otherwise, produce unspecified results.

3 arguments:
- If $2 is a binary primary, perform the binary test of $1 and $3.
- If $1 is `'!'`, negate the two-argument test of $2 and $3.
- Otherwise, produce unspecified results.

4 arguments:
- If $1 is `'!'`, negate the three-argument test of $2, $3, and $4.
- Otherwise, the results are unspecified.

>4 arguments: The results are unspecified.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *test*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of the **>** and **<** string comparison operators.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 *expression* evaluated to true.

1 *expression* evaluated to false or *expression* was missing.

>1 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Since `'>'` and `'<'` are operators in the shell language, applications need to quote them when passing them as arguments to *test* from a shell.

The **−a** and **−o** binary primaries and the `'('` and `')'` operators have been removed. (Many expressions using them were ambiguously defined by the grammar depending on the specific expressions being evaluated.) Scripts using these expressions should be converted to the forms given below. Even though many implementations will continue to support these forms, scripts should be extremely careful when dealing with user-supplied input that could be confused with these and other primaries and operators. Unless the application developer knows all the cases that produce input to the script, invocations like:

```
test "$1" -a "$2"
```

should be written as:

```
test "$1" && test "$2"
```

to avoid problems if a user supplied values such as $1 set to `'!'` and $2 set to the null string. That is, replace:

```
test expr1 -a expr2
```

with:

```
test expr1 && test expr2
```

and replace:

```
test expr1 -o expr2
```

with:

```
test expr1 || test expr2
```

but note that, in *test*, **−a** was specified as having higher precedence than **−o** while `"&&"` and `"||"` have equal precedence in the shell.

Parentheses or braces can be used in the shell command language to effect grouping.

The two commands:

```
test "$1"
test ! "$1"
```

could not be used reliably on some historical systems. Unexpected results would occur if such a *string* expression were used and $1 expanded to `'!'`, `'('`, or a known unary primary. Better constructs are:

```
test -n "$1"
test -z "$1"
```


117179 respectively.

117180 Historical systems have also been unreliable given the common construct:

117181 ```
test "$response" = "expected string"
```

117182 One of the following is a more reliable form:

117183 ```
test "X$response" = "Xexpected string"
```
117184 ```
test "expected string" = "$response"
```

117185 Note that the second form assumes that *expected string* could not be confused with any unary
117186 primary. If *expected string* starts with `'-'`, `'('`, `'!'`, or even `'='`, the first form should be used
117187 instead. Using the preceding rules, any of the three comparison forms is reliable, given any
117188 input. (However, note that the strings are quoted in all cases.)

117189 Historically, the string comparison binary primaries, `'='` and `"!="`, had a higher precedence
117190 than any unary primary in the greater than 4 argument case, and consequently unexpected
117191 results could occur if arguments were not properly prepared. For example, in:

117192 ```
test -d "$1" -o -d "$2"
```

117193 If $1 evaluates to a possible directory name of `'='`, the first three arguments are considered a
117194 string comparison, which causes a syntax error when the second **−d** is encountered. The
117195 following form prevents this:

117196 ```
test -d "$1" || test -d "$2"
```

117197 Also in the greater than 4 argument case:

117198 ```
test "$1" = "bat" -a "$2" = "ball"
```

117199 syntax errors would occur if $1 evaluates to `'('` or `'!'`. One of the following forms prevents
117200 this; the second is preferred:

117201 ```
test "$1" = "bat" && test "$2" = "ball"
```
117202 ```
test "X$1" = "Xbat" && test "X$2" = "Xball"
```

117203 Note that none of the following examples are permitted by the syntax described:

117204 ```
[-f file]
```
117205 ```
[-f file ]
```
117206 ```
[-f file]
```
117207 ```
[ -f file
```
117208 ```
test -f file]
```

117209 In the first two cases, if a utility named *[-f* exists, that utility would be invoked, and not *test*. In
117210 the remaining cases, the brackets are mismatched, and the behavior is unspecified. However:

117211 ```
test ! ]
```

117212 does have a defined meaning, and must exit with status 1. Similarly:

117213 ```
test]
```

117214 must exit with status 0.

117215 **EXAMPLES**

117216 1. Exit if there are not two or three arguments (two variations):

117217 ```
   if [ $# -ne 2 ] && [ $# -ne 3 ]; then exit 1; fi
```
117218 ```
   if [$# -lt 2] || [$# -gt 3]; then exit 1; fi
```

  
```

2. Perform a *mkdir* if a directory does not exist:

```
test ! -d tempdir && mkdir tempdir
```

3. Wait for a file to become non-readable:

```
while test -r thefile
do
    sleep 30
done
echo '"thefile" is no longer readable'
```

4. Perform a command if the argument is one of three strings (two variations):

```
if [ "$1" = "pear" ] || [ "$1" = "grape" ] || [ "$1" = "apple" ]
then
    command
fi
```

```
case "$1" in
    pear|grape|apple) command ;;
esac
```

RATIONALE

The KornShell-derived conditional command (double bracket **[[ ]]**) was removed from the shell command language description in an early proposal. Objections were raised that the real problem is misuse of the *test* command (**[**), and putting it into the shell is the wrong way to fix the problem. Instead, proper documentation and a new shell reserved word (**!**) are sufficient. A later proposal to add **[[ ]]** in Issue 8 was also rejected because existing implementations of it were found to be error-prone in a similar way to historical versions of *test*, and there was also too much variation in behavior between shells that support it.

Tests that require multiple *test* operations can be done at the shell level using individual invocations of the *test* command and shell logicals, rather than using the error-prone historical **−a** and **−o** operators of *test*.

The BSD and System V versions of **−f** were not the same. The BSD definition was:

−f *file* True if *file* exists and is not a directory.

The SVID version (true if the file exists and is a regular file) was chosen for this volume of POSIX.1-2024 because its use is consistent with the **−b**, **−c**, **−d**, and **−p** operands (*file* exists and is a specific file type).

The **−e** primary, possessing similar functionality to that provided by the C shell, was added because it provides the only way for a shell script to find out if a file exists without trying to open the file. Since implementations are allowed to add additional file types, a portable script cannot use:

```
test -b foo || test -c foo || test -d foo || test -f foo || test -p foo
```

to find out if **foo** is an existing file. On historical BSD systems, the existence of a file could be determined by:

```
test -f foo || test -d foo
```

but there was no easy way to determine that an existing file was a regular file. An early proposal used the KornShell **−a** primary (with the same meaning), but this was changed to **−e** because there were concerns about the high probability of humans confusing the **−a** primary with the historical **−a** binary operator.

The following options were not included in this volume of POSIX.1-2024, although they are provided by some implementations. These operands should not be used by new implementations for other purposes:

–k *file* True if *file* exists and its sticky bit is set.

–C *file* True if *file* is a contiguous file.

–V *file* True if *file* is a version file.

The following option was not included because it was undocumented in most implementations, has been removed from some implementations (including System V), and the functionality is provided by the shell (see Section 2.6.2 (on page 2485).

–l *string* The length of the string *string*.

The **–b**, **–c**, **–g**, **–p**, **–u**, and **–x** operands are derived from the SVID; historical BSD does not provide them. The **–k** operand is derived from System V; historical BSD does not provide it.

On historical BSD systems, *test* **–w** *directory* always returned false because *test* tried to open the directory for writing, which always fails.

Some additional primaries newly invented or from the KornShell appeared in an early proposal as part of the conditional command ([[ ]]): *s1* **>** *s2*, *s1* **<** *s2*, *f1* **–nt** *f2*, *f1* **–ot** *f2*, and *f1* **–ef** *f2*. They were not carried forward into the *test* utility when the conditional command was removed from the shell because they had not been included in the *test* utility built into historical implementations of the *sh* utility. However, they were later added to this standard once support for them became widespread.

The **–t** *file_descriptor* primary is shown with a mandatory argument because the grammar is ambiguous if it can be omitted. Historical implementations have allowed it to be omitted, providing a default of 1.

It is noted that `'['` is not part of the portable filename character set; however, since it is required to be encoded by a single byte, and is part of the portable character set, the name of this utility forms a character string across all supported locales.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 1.1.1.4 (on page 2454), *find*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

The **–h** operand is added for symbolic links, and access permission requirements are clarified for the **–r**, **–w**, and **–x** operands to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

The **–L** and **–S** operands are added for symbolic links and sockets.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/38 is applied, adding XSI margin marking and shading to a line in the OPERANDS section referring to the use of parentheses as

arguments to the *test* utility.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/30 is applied, rewording the existence primaries for the *test* utility.

Issue 7

Austin Group Interpretation 1003.1-2001 #107 is applied.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0143 [291] is applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0191 [898], XCU/TC2-2008/0192 [730], and XCU/TC2-2008/0193 [898] are applied.

Issue 8

Austin Group Defect 375 is applied, adding the *pathname1* **−ef** *pathname2*, *pathname1* **−nt** *pathname2*, *pathname1* **−ot** *pathname2*, *s1* **>** *s2*, and *s1* **<** *s2* primaries.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1330 is applied, removing the obsolescent (and optional) **−a** and **−o** binary primaries, and `'('` and `')'` operators.

Austin Group Defect 1348 is applied, removing ``()'' from ``the *exec*( ) family of functions''.

Austin Group Defect 1373 is applied, clarifying that when the utility name used is *[* the utility does not conform to Guidelines 1 and 2.

NAME

time — time a simple command

SYNOPSIS

```
time [-p] utility [argument...]
```

DESCRIPTION

The *time* utility shall invoke the utility named by the *utility* operand with arguments supplied as the *argument* operands and write a message to standard error that lists timing statistics for the utility. The message shall include the following information:

- The elapsed (real) time between invocation of *utility* and its termination.
- The User CPU time, equivalent to the sum of the *tms_utime* and *tms_cutime* fields returned by the *times*( ) function defined in the System Interfaces volume of POSIX.1-2024 for the process in which *utility* is executed.
- The System CPU time, equivalent to the sum of the *tms_stime* and *tms_cstime* fields returned by the *times*( ) function for the process in which *utility* is executed.

The precision of the timing shall be no less than the granularity defined for the size of the clock tick unit on the system, but the results shall be reported in terms of standard time units (for example, 0.02 seconds, 00:00:00.02, 1m33.75s, 365.21 seconds), not numbers of clock ticks.

When *time* is used in any of the following circumstances, via a simple command for which the word **time** is the command name (see Section 2.9.1.1, on page 2500), and none of the characters in the word **time** is quoted, the results (including parsing of later words) are unspecified:

- The simple command for which the word **time** is the command name includes one or more redirections (see Section 2.7, on page 2493) or is (directly) part of a pipeline (see Section 2.9.2, on page 2504).
- The next word that follows **time** would, if the word **time** were not present, be recognized as a reserved word (see Section 2.4, on page 2478) or a control operator (see XBD Section 3.85, on page 44).

Since these limitations only apply when *time* is executed via a simple command for which the word **time** is the command name and none of the characters in the word **time** is quoted, they can be avoided by quoting all or part of the word **time**, by arranging for the command name not to be **time** (for example, by having the command name be a word expansion), or by executing *time* via another utility such as *command* or *env*.

The limitations on redirections and pipelines can also be overcome by embedding the simple command within a compound command—most commonly a grouping command (see Section 2.9.4.1, on page 2508)—and applying the redirections or piping to the compound command instead.

Note that in no circumstances where the results are specified is it possible to apply different redirections to the *time* utility than are applied to the utility it invokes.

The following examples (where *a* and *b* are assumed to be the names of utilities found by searching *PATH*) show unspecified usages:

```
time a arg1 arg2 | b    # part of a pipeline
a | time -p b           # part of a pipeline
time a >/dev/null       # output redirection
</dev/null time a       # input redirection
time while anything...  # reserved word after time
time ( cmd )            # control operator after time
```



```
time;                   # control operator after time
time shift              # special built-in utility
time -p cd /            # intrinsic utility
```

The following examples have specified results and can be used as alternatives for the first four of the above when the *time* utility as specified here is intended to be invoked:

```
{ time a arg1 arg2; } | b
t=time; a | $t -p b
command time a >/dev/null
</dev/null \time a
```

OPTIONS

The *time* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

−p Write the timing output to standard error in the format shown in the STDERR section.

OPERANDS

The following operands shall be supported:

utility The name of a utility that is to be invoked. If the *utility* operand names a special built-in utility (see Section 2.15, on page 2526), an intrinsic utility (see Section 1.7, on page 2470), or a function (see Section 2.9.5, on page 2511), the results are unspecified.

argument Any string to be supplied as an argument when invoking the utility named by the *utility* operand.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *time*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic and informative messages written to standard error.

LC_NUMERIC
Determine the locale for numeric formatting.

117408 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

117409 *PATH* Determine the search path that shall be used to locate the utility to be invoked; see
117410 XBD Chapter 8 (on page 167).

117411 **ASYNCHRONOUS EVENTS**
117412 Default.

117413 **STDOUT**
117414 Not used.

117415 **STDERR**
117416 If the *utility* utility is invoked, the standard error shall be used to write the timing statistics and
117417 may be used to write a diagnostic message if the utility terminates abnormally; otherwise, the
117418 standard error shall be used to write diagnostic messages and may also be used to write the
117419 timing statistics.

117420 If **–p** is specified, the following format shall be used for the timing statistics in the POSIX locale:

117421 `"real %f\nuser %f\nsys %f\n", <real seconds>, <user seconds>,`
117422 `<system seconds>`

117423 where each floating-point number shall be expressed in seconds. The precision used may be less
117424 than the default six digits of `%f`, but shall be sufficiently precise to accommodate the size of the
117425 clock tick on the system (for example, if there were 60 clock ticks per second, at least two digits
117426 shall follow the radix character). The number of digits following the radix character shall be no
117427 less than one, even if this always results in a trailing zero. The implementation may append
117428 white space and additional information following the format shown here. The implementation
117429 may also prepend a single empty line before the format shown here.

117430 **OUTPUT FILES**
117431 None.

117432 **EXTENDED DESCRIPTION**
117433 None.

117434 **EXIT STATUS**
117435 If the *utility* utility is invoked, the exit status of *time* shall be the exit status of *utility*; otherwise,
117436 the *time* utility shall exit with one of the following values:

117437 1-125 An error occurred in the *time* utility.

117438 126 The utility specified by *utility* was found but could not be invoked.

117439 127 The utility specified by *utility* could not be found.

117440 **CONSEQUENCES OF ERRORS**
117441 Default.

117442 **APPLICATION USAGE**
117443 The *command*, *env*, *nice*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit
117444 code 127 if a utility to be invoked cannot be found, so that applications can distinguish ``failure
117445 to find a utility'' from ``invoked utility exited with an error indication''. The value 127 was
117446 chosen because it is not commonly used for other meanings; most utilities use small values for
117447 ``normal error conditions'' and the values above 128 can be confused with termination due to
117448 receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could
117449 be found, but not invoked. Some scripts produce meaningful error messages differentiating the
117450 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice
117451 that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any

attempt to *exec* the utility fails for any other reason.

EXAMPLES

It is frequently desirable to apply *time* to pipelines or lists of commands. This can be done by placing pipelines and command lists in a single file; this file can then be invoked as a utility, and the *time* applies to everything in the file.

Alternatively, the following command can be used to apply *time* to a complex command:

```
time sh -c -- 'complex-command-line'
```

RATIONALE

When the *time* utility was originally proposed to be included in the ISO POSIX-2: 1993 standard, questions were raised about its suitability for inclusion on the grounds that it was not useful for conforming applications, specifically:

- The underlying CPU definitions from the System Interfaces volume of POSIX.1-2024 are vague, so the numeric output could not be compared accurately between systems or even between invocations.
- The creation of portable benchmark programs was outside the scope this volume of POSIX.1-2024.

However, *time* does fit in the scope of user portability. Human judgement can be applied to the analysis of the output, and it could be very useful in hands-on debugging of applications or in providing subjective measures of system performance. Hence it has been included in this volume of POSIX.1-2024.

The default output format has been left unspecified because historical implementations differ greatly in their style of depicting this numeric output. The **−p** option was invented to provide scripts with a common means of obtaining this information.

In the KornShell, *time* is a shell reserved word that can be used to time an entire pipeline, rather than just a simple command. The POSIX definition has been worded to allow this implementation. Consideration was given to invalidating this approach because of the historical model from the C shell and System V shell. However, since the System V *time* utility historically has not produced accurate results in pipeline timing (because the constituent processes are not all owned by the same parent process, as allowed by POSIX), it did not seem worthwhile to break historical KornShell usage.

The term *utility* is used, rather than *command*, to highlight the fact that shell compound commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility* includes user application programs and shell scripts, not just the standard utilities.

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 2 (on page 2472), *sh*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *times*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is marked as part of the User Portability Utilities option.

Issue 7

The *time* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-115 is applied, updating the example in the DESCRIPTION.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0144 [266] is applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0194 [723] is applied.

Issue 8

Austin Group Defect 267 is applied, allowing **time** to be a reserved word.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1530 is applied, changing "`sh -c`" to "`sh -c --`".

Austin Group Defect 1586 is applied, adding the *timeout* utility.

Austin Group Defect 1594 is applied, changing the APPLICATION USAGE section.

NAME

timeout — execute a utility with a time limit

SYNOPSIS

```
timeout [-fp] [-k time] [-s signal_name] duration utility [argument...]
```

DESCRIPTION

The *timeout* utility shall execute the utility named by the *utility* operand, with arguments supplied as the *argument* operands (if any), in a child process. If the value of the *duration* operand is non-zero and the child process has not terminated after the specified time period, *timeout* shall send the signal specified by the **–s** option, or the SIGTERM signal if **–s** is not given.

If the **–f** option is specified, the signal shall be sent only to the child process. Otherwise, it is implementation defined which one of the following methods is used to signal additional processes:

- The *timeout* utility ensures it is a process group leader before creating the child process which executes the utility, in which case it shall send the signal to its process group.
- The *timeout* utility arranges for any descendants of the child process that are orphaned to have their parent process changed to the *timeout* utility, in which case the signal shall be sent to the child process and all of its descendants.

If the subsequent wait status of the child process shows that it was stopped by a signal, a SIGCONT signal shall also be sent in the same manner as the first signal; otherwise, a SIGCONT signal may be sent in the same manner.

If the **–k** option is specified, and the child process created to execute the utility still has not terminated after the time period specified by the *time* option-argument has elapsed since the first signal was sent, *timeout* shall send a SIGKILL signal in the same manner as the first signal. If *timeout* receives a signal and propagates it to the child process (see ASYNCHRONOUS EVENTS below), this shall be treated as the first signal.

OPTIONS

The *timeout* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–f Only time out the utility itself, not its descendants.

–k *time* Send a SIGKILL signal if the child process created to execute the utility has not terminated after the time period specified by *time* has elapsed since the first signal was sent. The value of *time* shall be interpreted as specified for the *duration* operand (see OPERANDS below).

–p Always preserve (mimic) the wait status of the executed utility, even if the time limit was reached.

–s *signal_name*
Specify the signal to send when the time limit is reached, using one of the symbolic names defined in the **<signal.h>** header. Values of *signal_name* shall be recognized in a case-independent fashion, without the SIG prefix. By default, SIGTERM shall be sent.

OPERANDS

The following operands shall be supported:

duration The maximum amount of time to allow the utility to run, specified as a decimal number with an optional decimal fraction and an optional suffix, which can be:

s seconds

m minutes

h hours

d days

If a decimal fraction is present, the application shall ensure that it is separated from the units by a <period>. If no suffix is present, the value shall specify seconds.

If the value is zero, *timeout* shall not enforce a time limit.

utility The name of a utility that is to be executed. If the *utility* operand names any of the special built-in utilities in Section 2.15 (on page 2526), the results are undefined.

argument Any string to be supplied as an argument when executing the utility named by the *utility* operand.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *timeout*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

PATH Determine the search path that is used to locate the utility to be executed. See XBD Section 8.3 (on page 174).

ASYNCHRONOUS EVENTS

The default behavior specified in Section 1.4 (on page 2462) shall apply, except that:

- The *timeout* utility shall ignore SIGTTIN and SIGTTOU signals.
- The *timeout* utility may alter the disposition of SIGALRM if the inherited disposition was for it to be ignored.
- If the signal specified with the **–s** option, or any signal whose default action is to terminate the process, is delivered to the *timeout* utility, then unless the signal is SIGKILL or SIGSTOP, the *timeout* utility shall immediately send the same signal to the process or processes to which it would send a signal when the time limit is reached. If the delivered signal is SIGALRM, *timeout* may behave as if the time limit had been reached instead of sending SIGALRM.

- If the **−f** option is not specified, then if *timeout* sends a signal to its process group, it shall briefly change the disposition of that signal to ignored while it sends the signal, so that it does not receive the signal itself.

With the single exception of the signal specified with the **−s** option, or SIGTERM if **−s** is not used, all signal dispositions inherited by the utility specified by the *utility* operand shall be the same as the disposition that *timeout* inherited.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

If the **−p** option is not specified and the time limit was reached:

- If the **−k** option was not specified or the utility terminated before the time period specified by the *time* option-argument elapsed since the first signal was sent, the exit status shall be 124.
- If the **−k** option was specified and the SIGKILL signal was sent, it is unspecified whether the exit status is 124 or the behavior is as if the **−p** option was specified.

Otherwise, if the executed utility terminated by exiting, the exit status of *timeout* shall be that of the utility; if the utility was terminated by a signal, *timeout* shall terminate itself with the same signal while ensuring that a core image is not created.

If an error occurs, the following exit values shall be returned:

125 An error other than the two described below occurred.

126 The utility specified by *utility* was found but could not be executed.

127 The utility specified by *utility* could not be found.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Unlike the *kill* utility, the **−s** option of *timeout* is not required to accept the symbolic name 0 to represent signal value zero.

When the value of *duration* is zero, *timeout* does not time out the utility, but it does still perform signal propagation (including to descendants of the utility if **−f** is not specified).

Regardless of locale, the <period> character (the decimal-point character of the POSIX locale) is the decimal-point character recognized in the *duration* operand and the *time* option-argument.

The *command*, *env*, *nice*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit code 127 if a utility to be invoked cannot be found, so that applications can distinguish ``failure to find a utility'' from ``invoked utility exited with an error indication''. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for ``normal error conditions'' and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could

be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for any other reason. The *timeout* utility extends these special exit codes to 125 and 124, with the meanings described in EXIT STATUS. A *timeout* exit status below 124 can only result from passing through the exit status of the executed utility.

EXAMPLES

None.

RATIONALE

Some *timeout* implementations make themselves a process group leader (when **−f** is not used) in order to be able to send signals to descendants of the child process. However, using this method means that any descendants which change their process group do not receive the signal. To ensure all descendants receive the signal, some implementations instead make use of a feature whereby descendants that are orphaned have their parent process changed to the *timeout* utility—that is, *timeout* becomes their ``reaper''—together with the ability of a reaper to send a signal to all of its descendants.

Some historical *timeout* implementations exited with status 128+*signal_number* when the child process was terminated by a signal before the time limit was reached (or when **−p** was used). This is reasonable when *timeout* is invoked from a shell which sets $? to 128+*signal_number*, but not all shells do that. In particular, the KornShell sets $? to 256+*signal_number* and so an exit status of 128+*signal_number* from *timeout* would be misleading. In order to avoid any possible ambiguity, this standard requires that *timeout* mimics the wait status of the child process by terminating itself with the same signal. When it does this it needs to ensure that it does not create a core image, otherwise it could overwrite one created by the invoked utility.

The *timeout* utility ignores SIGTTIN and SIGTTOU so that if the utility it executes reads from or writes to the controlling terminal and this generates a SIGTTIN or SIGTTOU for the process group, *timeout* will not be stopped by the signal and can still time out the utility.

Some historical *timeout* implementations always set the disposition for SIGTTIN and SIGTTOU in the child process to default, even if these signals were inherited as ignored. This could result in processes being stopped unexpectedly. Likewise, they did not ensure that for signals they caught, the disposition inherited by the executed utility was the same as the disposition that was inherited by *timeout*. This meant that, for example, if *timeout* was used in a script that was run with *nohup*, the utility executed by *timeout* would unexpectedly not be protected from SIGHUP. This standard requires that all signal dispositions inherited by the utility specified by the *utility* operand are the same as the disposition that *timeout* inherited, with the single exception of the signal that *timeout* sends when the time limit is reached, which needs to be inherited as default in order for the timeout to take effect (without resorting to SIGKILL if **−k** is specified).

Some historical *timeout* implementations only propagated a subset of the signals whose default action is to terminate the process to the child process if one was delivered to the *timeout* utility. Propagating these signals is beneficial, as otherwise termination of the *timeout* utility by a signal results in the utility it executed being left running indefinitely (unless it also received the signal, for example a terminal-generated SIGINT). There is no reason to select a subset of these signals to be propagated, therefore this standard requires them all to be propagated (except SIGKILL, which cannot). In the event that a user wants to prevent the utility being timed out, sending *timeout* a SIGKILL can be used for this purpose.

FUTURE DIRECTIONS

None.

SEE ALSO

kill

XBD Chapter 8 (on page 167), Section 12.2 (on page 215), **<signal.h>** (on page 346)

CHANGE HISTORY

First released in Issue 8.

NAME

touch — change file access and modification times

SYNOPSIS

```
touch [-acm] [-r ref_file|-t time|-d date_time] file...
```

DESCRIPTION

The *touch* utility shall change the last data modification timestamps, the last data access timestamps, or both.

The time used can be specified by the **–t** *time* option-argument, the corresponding *time* fields of the file referenced by the **–r** *ref_file* option-argument, or the **–d** *date_time* option-argument, as specified in the following sections. If none of these are specified, *touch* shall use the current time.

For each *file* operand, *touch* shall perform actions equivalent to the following functions defined in the System Interfaces volume of POSIX.1-2024:

1. If *file* does not exist:
   a. The *creat*( ) function is called with the following arguments:
      — The *file* operand is used as the *path* argument.
      — The value of the bitwise-inclusive OR of S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH is used as the *mode* argument.
   b. The *futimens*( ) function is called with the following arguments:
      — The file descriptor opened in step 1a.
      — The access time and the modification time, set as described in the OPTIONS section, are used as the first and second elements of the *times* array argument, respectively.
2. If *file* exists, the *utimensat*( ) function is called with the following arguments:
   a. The AT_FDCWD special value is used as the *fd* argument.
   b. The *file* operand is used as the *path* argument.
   c. The access time and the modification time, set as described in the OPTIONS section, are used as the first and second elements of the *times* array argument, respectively.
   d. The *flag* argument is set to zero.

OPTIONS

The *touch* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–a Change the access time of *file*. Do not change the modification time unless **–m** is also specified.

–c Do not create a specified *file* if it does not exist. Do not write any diagnostic messages concerning this condition.

–d *date_time* Use the specified *date_time* instead of the current time. The option-argument shall be a string of the form:

*YYYY–MM–DD*T*hh*:*mm*:*SS*[.*frac*][*tz*]

or:

117728 *YYYY*–*MM*–*DD*T*hh*:*mm*:*SS*[,*frac*][*tz*]

117729 where:

117730 • *YYYY* are at least four decimal digits giving the year.

117731 • *MM*, *DD*, *hh*, *mm*, and *SS* are as with –**t** *time*.

117732 • T is the time designator, and can be replaced by a single <space>.

117733 • [.*frac*] and [,*frac*] are either empty, or a <period> ('.') or a <comma>
117734 (',') respectively, followed by one or more decimal digits, specifying a
117735 fractional second.

117736 • [*tz*] is either empty, signifying local time, or the letter 'Z', signifying UTC.
117737 If [*tz*] is empty, the resulting time shall be affected by the value of the *TZ*
117738 environment variable.

117739 If the resulting time precedes the Epoch, the behavior is implementation-defined. If
117740 the time cannot be represented as the file's timestamp, *touch* shall exit immediately
117741 with an error status.

117742 –**m** Change the modification time of *file*. Do not change the access time unless –**a** is
117743 also specified.

117744 –**r** *ref_file* Use the corresponding time of the file named by the pathname *ref_file* instead of
117745 the current time.

117746 –**t** *time* Use the specified *time* instead of the current time. The option-argument shall be a
117747 decimal number of the form:

117748 **[[***CC***]***YY***]***MMDDhhmm***[.***SS***]**

117749 where each two digits represents the following:

117750 *MM* The month of the year [01,12].

117751 *DD* The day of the month [01,31].

117752 *hh* The hour of the day [00,23].

117753 *mm* The minute of the hour [00,59].

117754 *CC* The first two digits of the year (the century).

117755 *YY* The second two digits of the year.

117756 *SS* The second of the minute [00,60].

117757 Both *CC* and *YY* shall be optional. If neither is given, the current year shall be
117758 assumed. If *YY* is specified, but *CC* is not, *CC* shall be derived as follows:

117759

If *YY* **is:**	*CC* **becomes:**
[69,99]	19
[00,68]	20

117760
117761

117762 **Note:** It is expected that in a future version of this standard the default century inferred
117763 from a 2-digit year will change. (This would apply to all commands accepting a
117764 2-digit year as input.)

117765 The resulting time shall be affected by the value of the *TZ* environment variable. If
117766 the resulting time value precedes the Epoch, the behavior is implementation-
117767 defined. If the time is out of range for the file's timestamp, *touch* shall exit

117768 immediately with an error status. The range of valid times past the Epoch is
117769 implementation-defined, but it shall extend to at least the time 0 hours, 0 minutes,
117770 0 seconds, January 1, 2038, Coordinated Universal Time. Some implementations
117771 may not be able to represent dates beyond January 18, 2038, because they use
117772 **signed int** as a time holder.

117773 The range for *SS* is [00,60] rather than [00,59] because of leap seconds. If *SS* is 60,
117774 and the resulting time, as affected by the *TZ* environment variable, does not refer
117775 to a leap second, the resulting time shall be one second after a time where *SS* is 59.
117776 If *SS* is not given a value, it is assumed to be zero.

117777 If neither the **−a** nor **−m** options were specified, *touch* shall behave as if both the **−a** and **−m**
117778 options were specified.

117779 **OPERANDS**

117780 The following operands shall be supported:

117781 *file* A pathname of a file whose times shall be modified.

117782 **STDIN**

117783 Not used.

117784 **INPUT FILES**

117785 None.

117786 **ENVIRONMENT VARIABLES**

117787 The following environment variables shall affect the execution of *touch*:

117788 *LANG* Provide a default value for the internationalization variables that are unset or null.
117789 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
117790 variables used to determine the values of locale categories.)

117791 *LC_ALL* If set to a non-empty string value, override the values of all the other
117792 internationalization variables.

117793 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
117794 characters (for example, single-byte as opposed to multi-byte characters in
117795 arguments).

117796 *LC_MESSAGES*
117797 Determine the locale that should be used to affect the format and contents of
117798 diagnostic messages written to standard error.

117799 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

117800 *TZ* Determine the timezone to be used for interpreting the *time* option-argument. If *TZ*
117801 is unset or null, an unspecified default timezone shall be used.

117802 **ASYNCHRONOUS EVENTS**

117803 Default.

117804 **STDOUT**

117805 Not used.

117806 **STDERR**

117807 The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 The utility executed successfully and all requested changes were made.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The interpretation of time is taken to be *seconds since the Epoch* (see XBD Section 4.19, on page 107). It should be noted that implementations conforming to the System Interfaces volume of POSIX.1-2024 do not take leap seconds into account when computing seconds since the Epoch. When *SS*=60 is used, the resulting time always refers to 1 plus *seconds since the Epoch* for a time when *SS*=59.

Although the **–t** *time* option-argument specifies values in 1969, the access time and modification time fields are defined in terms of seconds since the Epoch (00:00:00 on 1 January 1970 UTC). Therefore, depending on the value of *TZ* when *touch* is run, there is never more than a few valid hours in 1969 and there need not be any valid times in 1969.

If the *T* time designator is replaced by a <space> for the **–d** *date_time* option-argument, the <space> must be quoted to prevent the shell from splitting the argument.

EXAMPLES

Create or update a file called **dwc**; the resulting file has both the last data modification and last data access timestamps set to November 12, 2007 at 10:15:30 local time:

```
touch -d 2007-11-12T10:15:30 dwc
```

Create or update a file called **nick**; the resulting file has both the last data modification and last data access timestamps set to November 12, 2007 at 10:15:30 UTC:

```
touch -d 2007-11-12T10:15:30Z nick
```

Create or update a file called **gwc**; the resulting file has both the last data modification and last data access timestamps set to November 12, 2007 at 10:15:30 local time with a fractional second timestamp of .002 seconds:

```
touch -d 2007-11-12T10:15:30,002 gwc
```

Create or update a file called **ajosey**; the resulting file has both the last data modification and last data access timestamps set to November 12, 2007 at 10:15:30 UTC with a fractional second timestamp of .002 seconds:

```
touch -d "2007-11-12 10:15:30.002Z" ajosey
```

Create or update a file called **cathy**; the resulting file has both the last data modification and last data access timestamps set to November 12, 2007 at 10:15:00 local time:

```
touch -t 200711121015 cathy
```

Create or update a file called **drepper**; the resulting file has both the last data modification and last data access timestamps set to November 12, 2007 at 10:15:30 local time:


```
touch -t 200711121015.30 drepper
```

Create or update a file called **ebb9**; the resulting file has both the last data modification and last data access timestamps set to November 12, 2007 at 10:15:30 local time:

```
touch -t 0711121015.30 ebb9
```

Create or update a file called **eggert**; the resulting file has the last data access timestamp set to the corresponding time of the file named **mark** instead of the current time. If the file exists, the last data modification time is not changed:

```
touch -a -r mark eggert
```

RATIONALE

The functionality of *touch* is described almost entirely through references to functions in the System Interfaces volume of POSIX.1-2024. In this way, there is no duplication of effort required for describing such side-effects as the relationship of user IDs to the user database, permissions, and so on.

There are some significant differences between the *touch* utility in this volume of POSIX.1-2024 and those in System V and BSD systems. They are upwards-compatible for historical applications from both implementations:

1. In System V, an ambiguity exists when a pathname that is a decimal number leads the operands; it is treated as a time value. In BSD, no *time* value is allowed; files may only be *touch*ed to the current time. The **−t** *time* construct solves these problems for future conforming applications (note that the **−t** option is not historical practice).
2. The inclusion of the century digits, *CC*, is also new. Note that a ten-digit *time* value is treated as if *YY*, and not *CC*, were specified. The caveat about the range of dates following the Epoch was included as recognition that some implementations are not able to represent dates beyond 18 January 2038 because they use **signed int** as a time holder.

The **−r** option was added because several comments requested this capability. This option was named **−f** in an early proposal, but was changed because the **−f** option is used in the BSD version of *touch* with a different meaning.

At least one historical implementation of *touch* incremented the exit code if **−c** was specified and the file did not exist. This volume of POSIX.1-2024 requires exit status zero if no errors occur.

In previous version of the standard, if at least two operands are specified, and the first operand is an eight or ten-digit decimal integer, the first operand was assumed to be a *date_time* operand. This usage was removed in this version of the standard since it had been marked obsolescent previously.

The **−d** *date_time* format is an ISO 8601-1: 2019 standard complete representation of date and time extended format with an optional decimal point or <comma> followed by a string of digits following the seconds portion to specify fractions of a second. It is not necessary to recognize `"[+/-]hh:mm"` and `"[+/-]hh"` to specify timezones other than local time and UTC. The *T* time designator in the ISO 8601-1: 2019 standard extended format may be replaced by <space>.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

date

XBD Section 4.19 (on page 107), Chapter 8 (on page 167), Section 12.2 (on page 215), **<sys/stat.h>**

XSH *creat*( ), *futimens*( ), *time*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

The obsolescent *date_time* operand is removed.

The Open Group Corrigendum U027/1 is applied. This extends the range of valid time past the Epoch to at least the time 0 hours, 0 minutes, 0 seconds, January 1, 2038, Coordinated Universal Time. This is a new requirement on POSIX implementations.

The range for seconds is changed from [00,61] to [00,60] to align with the ISO/IEC 9899:1999 standard, and to allow for positive leap seconds.

Issue 7

Austin Group Interpretation 1003.1-2001 #118 is applied.

Austin Group Interpretation 1003.1-2001 #193 is applied, adding support for subsecond timestamps.

SD5-XCU-ERN-45 is applied, adding a new paragraph to the RATIONALE.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

SD5-XCU-ERN-110 is applied, updating the OPTIONS section.

Changes are made related to support for finegrained timestamps.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0195 [474] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

tput — change terminal characteristics

SYNOPSIS

```
tput [-T type] operand...
```

DESCRIPTION

The *tput* utility shall display terminal-dependent information. The manner in which this information is retrieved is unspecified. The information displayed shall clear the terminal screen, initialize the user's terminal, or reset the user's terminal, depending on the operand given. The exact consequences of displaying this information are unspecified.

OPTIONS

The *tput* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

–T *type* Indicate the type of terminal. If this option is not supplied and the *TERM* variable is unset or null, an unspecified default terminal type shall be used. The setting of *type* shall take precedence over the value in *TERM*.

OPERANDS

The following strings shall be supported as operands by the implementation in the POSIX locale:

clear Display the clear-screen sequence.

init Display the sequence that initializes the user's terminal in an implementation-defined manner.

reset Display the sequence that resets the user's terminal in an implementation-defined manner.

If a terminal does not support any of the operations described by these operands, this shall not be considered an error condition.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *tput*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

117959 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

117960 *TERM* Determine the terminal type. If this variable is unset or null, and if the **–T** option is
117961 not specified, an unspecified default terminal type shall be used.

117962 **ASYNCHRONOUS EVENTS**
117963 Default.

117964 **STDOUT**
117965 If standard output is a terminal device, it may be used for writing the appropriate sequence to
117966 clear the screen or reset or initialize the terminal. If standard output is not a terminal device,
117967 undefined results occur.

117968 **STDERR**
117969 The standard error shall be used only for diagnostic messages.

117970 **OUTPUT FILES**
117971 None.

117972 **EXTENDED DESCRIPTION**
117973 None.

117974 **EXIT STATUS**
117975 The following exit values shall be returned:

117976 0 The requested string was written successfully.

117977 1 Unspecified.

117978 2 Usage error.

117979 3 No information is available about the specified terminal type.

117980 4 The specified operand is invalid.

117981 >4 An error occurred.

117982 **CONSEQUENCES OF ERRORS**
117983 If one of the operands is not available for the terminal, *tput* continues processing the remaining
117984 operands.

117985 **APPLICATION USAGE**
117986 The difference between resetting and initializing a terminal is left unspecified, as they vary
117987 greatly based on hardware types. In general, resetting is a more severe action.

117988 Some terminals use control characters to perform the stated functions, and on such terminals it
117989 might make sense to use *tput* to store the initialization strings in a file or environment variable
117990 for later use. However, because other terminals might rely on system calls to do this work, the
117991 standard output cannot be used in a portable manner, such as the following non-portable
117992 constructs:

```
117993 ClearVar=`tput clear`
117994 tput reset | mailx -s "Wake Up" ddg
```

117995 **EXAMPLES**

117996 1. Initialize the terminal according to the type of terminal in the environmental variable
117997 *TERM*. This command can be included in a **.profile** file.

```
117998 tput init
```


2. Reset a 450 terminal.

```
tput -T 450 reset
```

RATIONALE

The list of operands was reduced to a minimum for the following reasons:

- The only features chosen were those that were likely to be used by human users interacting with a terminal.
- Specifying the full *terminfo* set was not considered desirable, but the standard developers did not want to select among operands.
- This volume of POSIX.1-2024 does not attempt to provide applications with sophisticated terminal handling capabilities, as that falls outside of its assigned scope and intersects with the responsibilities of other standards bodies.

The difference between resetting and initializing a terminal is left unspecified as this varies greatly based on hardware types. In general, resetting is a more severe action.

The exit status of 1 is historically reserved for finding out if a Boolean operand is not set. Although the operands were reduced to a minimum, the exit status of 1 should still be reserved for the Boolean operands, for those sites that wish to support them.

FUTURE DIRECTIONS

None.

SEE ALSO

stty, *tabs*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is marked as part of the User Portability Utilities option.

Issue 7

The *tput* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

tr — translate characters

SYNOPSIS

```
tr [-c|-C] [-s] string1 string2
tr -s [-c|-C] string1
tr -d [-c|-C] string1
tr -ds [-c|-C] string1 string2
```

DESCRIPTION

The *tr* utility shall copy the standard input to the standard output with substitution or deletion of selected characters. The options specified and the *string1* and *string2* operands shall control translations that occur while copying characters and single-character collating elements.

OPTIONS

The *tr* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−c Complement the set of values specified by *string1*. See the EXTENDED DESCRIPTION section.

−C Complement the set of characters specified by *string1*. See the EXTENDED DESCRIPTION section.

−d Delete all occurrences of input characters that are specified by *string1*.

−s Replace instances of repeated characters with a single character, as described in the EXTENDED DESCRIPTION section.

OPERANDS

The following operands shall be supported:

string1, *string2*
Translation control strings. Each string shall represent a set of characters to be converted into an array of characters used for the translation. For a detailed description of how the strings are interpreted, see the EXTENDED DESCRIPTION section.

STDIN

The standard input can be any type of file.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *tr*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of range expressions and equivalence classes.

118070 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
118071 characters (for example, single-byte as opposed to multi-byte characters in
118072 arguments) and the behavior of character classes.

118073 *LC_MESSAGES*
118074 Determine the locale that should be used to affect the format and contents of
118075 diagnostic messages written to standard error.

118076 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

118077 **ASYNCHRONOUS EVENTS**
118078 Default.

118079 **STDOUT**
118080 The *tr* output shall be identical to the input, with the exception of the specified transformations.

118081 **STDERR**
118082 The standard error shall be used only for diagnostic messages.

118083 **OUTPUT FILES**
118084 None.

118085 **EXTENDED DESCRIPTION**
118086 The operands *string1* and *string2* (if specified) define two arrays of characters. The constructs in
118087 the following list can be used to specify characters or single-character collating elements. If any
118088 of the constructs result in multi-character collating elements, *tr* shall exclude, without a
118089 diagnostic, those multi-character elements from the resulting array.

118090 *character* Any character not described by one of the conventions below shall represent itself.

118091 *\octal* Octal sequences can be used to represent characters with specific coded values. An
118092 octal sequence shall consist of a <backslash> followed by the longest sequence of
118093 one, two, or three-octal-digit characters (01234567). The sequence shall cause the
118094 value whose encoding is represented by the one, two, or three-digit octal integer to
118095 be placed into the array. Multi-byte characters require multiple, concatenated
118096 escape sequences of this type, including the leading <backslash> for each byte.

118097 *\character* The <backslash>-escape sequences in XBD Table 5-1 (on page 113) (`'\\'`, `'\a'`,
118098 `'\b'`,`'\f'`,`'\n'`,`'\r'`,`'\t'`,`'\v'`) shall be supported. The results of using any
118099 other character, other than an octal digit, following the <backslash> are
118100 unspecified. Also, if there is no character following the <backslash>, the results are
118101 unspecified.

118102 *c–c* In the POSIX locale, this construct shall represent the range of collating elements
118103 between the range endpoints (as long as neither endpoint is an octal sequence of
118104 the form *\octal*), inclusive, as defined by the collation sequence. The characters or
118105 collating elements in the range shall be placed in the array in ascending collation
118106 sequence. If the second endpoint precedes the starting endpoint in the collation
118107 sequence, it is unspecified whether the range of collating elements is empty, or this
118108 construct is treated as invalid. In locales other than the POSIX locale, this construct
118109 has unspecified behavior.

118110 If either or both of the range endpoints are octal sequences of the form *\octal*, this
118111 shall represent the range of specific coded values between the two range
118112 endpoints, inclusive.

[:*class*:] Represents all characters belonging to the defined character class, as defined by the current setting of the *LC_CTYPE* locale category. The following character class names shall be accepted when specified in *string1*:

alnum	**blank**	**digit**	**lower**	**punct**	**upper**
alpha	**cntrl**	**graph**	**print**	**space**	**xdigit**

XSI In addition, character class expressions of the form [:*name*:] shall be recognized in those locales where the *name* keyword has been given a **charclass** definition in the *LC_CTYPE* category.

When both the −**d** and −**s** options are specified, any of the character class names shall be accepted in *string2*. Otherwise, only character class names **lower** or **upper** are valid in *string2* and then only if the corresponding character class (**upper** and **lower**, respectively) is specified in the same relative position in *string1*. Such a specification shall be interpreted as a request for case conversion. When [:*lower*:] appears in *string1* and [:*upper*:] appears in *string2*, the arrays shall contain the characters from the **toupper** mapping in the *LC_CTYPE* category of the current locale. When [:*upper*:] appears in *string1* and [:*lower*:] appears in *string2*, the arrays shall contain the characters from the **tolower** mapping in the *LC_CTYPE* category of the current locale. The first character from each mapping pair shall be in the array for *string1* and the second character from each mapping pair shall be in the array for *string2* in the same relative position.

Except for case conversion, the characters specified by a character class expression shall be placed in the array in an unspecified order.

If the name specified for *class* does not define a valid character class in the current locale, the behavior is undefined.

[=*equiv*=] Represents all characters or collating elements belonging to the same equivalence class as *equiv*, as defined by the current setting of the *LC_COLLATE* locale category. An equivalence class expression shall be allowed only in *string1*, or in *string2* when it is being used by the combined −**d** and −**s** options. The characters belonging to the equivalence class shall be placed in the array in an unspecified order.

[*x***n*] Represents *n* repeated occurrences of the character *x*. Because this expression is used to map multiple characters to one, it is only valid when it occurs in *string2*. If *n* is omitted or is zero, it shall be interpreted as large enough to extend the *string2*-based sequence to the length of the *string1*-based sequence. If *n* has a leading zero, it shall be interpreted as an octal value. Otherwise, it shall be interpreted as a decimal value.

When the −**d** option is not specified:

- If *string2* is present, each input character found in the array specified by *string1* shall be replaced by the character in the same relative position in the array specified by *string2*. If the array specified by *string2* is shorter that the one specified by *string1*, or if a character occurs more than once in *string1*, the results are unspecified.
- If the −**C** option is specified, the complements of the characters specified by *string1* (the set of all characters in the current character set, as defined by the current setting of *LC_CTYPE*, except for those actually specified in the *string1* operand) shall be placed in the array in ascending collation sequence, as defined by the current setting of *LC_COLLATE*.

- If the **−c** option is specified, the complement of the values specified by *string1* shall be placed in the array in ascending order by binary value.
- Because the order in which characters specified by character class expressions or equivalence class expressions is undefined, such expressions should only be used if the intent is to map several characters into one. An exception is case conversion, as described previously.

When the **−d** option is specified:

- Input characters found in the array specified by *string1* shall be deleted.
- When the **−C** option is specified with **−d**, all characters except those specified by *string1* shall be deleted. The contents of *string2* are ignored, unless the **−s** option is also specified.
- When the **−c** option is specified with **−d**, all values except those specified by *string1* shall be deleted. The contents of *string2* shall be ignored, unless the **−s** option is also specified.
- The same string cannot be used for both the **−d** and the **−s** option; when both options are specified, both *string1* (used for deletion) and *string2* (used for squeezing) shall be required.

When the **−s** option is specified, after any deletions or translations have taken place, repeated sequences of the same character shall be replaced by one occurrence of the same character, if the character is found in the array specified by the last operand. If the last operand contains a character class, such as the following example:

```
tr -s '[:space:]'
```

the last operand's array shall contain all of the characters in that character class. However, in a case conversion, as described previously, such as:

```
tr -s '[:upper:]' '[:lower:]'
```

the last operand's array shall contain only those characters defined as the second characters in each of the **toupper** or **tolower** character pairs, as appropriate.

An empty string used for *string1* or *string2* produces undefined results.

EXIT STATUS

The following exit values shall be returned:

0 All input was processed successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

If necessary, *string1* and *string2* can be quoted to avoid pattern matching by the shell.

If an ordinary digit (representing itself) is to follow an octal sequence, the octal sequence must use the full three digits to avoid ambiguity.

When *string2* is shorter than *string1*, a difference results between historical System V and BSD systems. A BSD system pads *string2* with the last character found in *string2*. Thus, it is possible to do the following:

```
tr 0123456789 d
```

which would translate all digits to the letter `'d'`. Since this area is specifically unspecified in

this volume of POSIX.1-2024, both the BSD and System V behaviors are allowed, but a conforming application cannot rely on the BSD behavior. It would have to code the example in the following way:

```
tr 0123456789 '[d*]'
```

It should be noted that, despite similarities in appearance, the string operands used by *tr* are not regular expressions.

Unlike some historical implementations, this definition of the *tr* utility correctly processes NUL characters in its input stream. NUL characters can be stripped by using:

```
tr -d '\000'
```

EXAMPLES

1. The following example creates a list of all words in **file1** one per line in **file2**, where a word is taken to be a maximal string of letters.

   ```
   tr -cs "[:alpha:]" "[\n*]" <file1 >file2
   ```

2. The next example translates all lowercase characters in **file1** to uppercase and writes the results to standard output.

   ```
   tr "[:lower:]" "[:upper:]" <file1
   ```

3. This example uses an equivalence class to identify accented variants of the base character `'e'` in **file1**, which are stripped of diacritical marks and written to **file2**.

   ```
   tr "[=e=]" "[e*]" <file1 >file2
   ```

RATIONALE

In some early proposals, an explicit option **–n** was added to disable the historical behavior of stripping NUL characters from the input. It was considered that automatically stripping NUL characters from the input was not correct functionality. However, the removal of **–n** in a later proposal does not remove the requirement that *tr* correctly process NUL characters in its input stream. NUL characters can be stripped by using *tr* **–d** '\000'.

Historical implementations of *tr* differ widely in syntax and behavior. For example, the BSD version has not needed the bracket characters for the repetition sequence. The *tr* utility syntax is based more closely on the System V and XPG3 model while attempting to accommodate historical BSD implementations. In the case of the short *string2* padding, the decision was to unspecify the behavior and preserve System V and XPG3 scripts, which might find difficulty with the BSD method. The assumption was made that BSD users of *tr* have to make accommodations to meet the syntax defined here. Since it is possible to use the repetition sequence to duplicate the desired behavior, whereas there is no simple way to achieve the System V method, this was the correct, if not desirable, approach.

The use of octal values to specify control characters, while having historical precedents, is not portable. The introduction of escape sequences for control characters should provide the necessary portability. It is recognized that this may cause some historical scripts to break.

An early proposal included support for multi-character collating elements. It was pointed out that, while *tr* does employ some syntactical elements from REs, the aim of *tr* is quite different; ranges, for example, do not have a similar meaning (``any of the chars in the range matches'', *versus* ``translate each character in the range to the output counterpart''). As a result, the previously included support for multi-character collating elements has been removed. What remains are ranges in current collation order (to support, for example, accented characters), character classes, and equivalence classes.

In XPG3 the [:*class*:] and [=*equiv*=] conventions are shown with double brackets, as in RE syntax. However, *tr* does not implement RE principles; it just borrows part of the syntax. Consequently, [:*class*:] and [=*equiv*=] should be regarded as syntactical elements on a par with [*x***n*], which is not an RE bracket expression.

The standard developers will consider changes to *tr* that allow it to translate characters between different character encodings, or they will consider providing a new utility to accomplish this.

On historical System V systems, a range expression requires enclosing square-brackets, such as:

```
tr '[a-z]' '[A-Z]'
```

However, BSD-based systems did not require the brackets, and this convention is used here to avoid breaking large numbers of BSD scripts:

```
tr a-z A-Z
```

The preceding System V script will continue to work because the brackets, treated as regular characters, are translated to themselves. However, any System V script that relied on `"a-z"` representing the three characters `'a'`, `'-'`, and `'z'` have to be rewritten as `"az-"`.

The ISO POSIX-2: 1993 standard had a **−c** option that behaved similarly to the **−C** option, but did not supply functionality equivalent to the **−c** option specified in POSIX.1-2024.

The earlier version also said that octal sequences referred to collating elements and could be placed adjacent to each other to specify multi-byte characters. However, it was noted that this caused ambiguities because *tr* would not be able to tell whether adjacent octal sequences were intending to specify multi-byte characters or multiple single byte characters. POSIX.1-2024 specifies that octal sequences always refer to single byte binary values when used to specify an endpoint of a range of collating elements.

Earlier versions of this standard allowed for implementations with bytes other than eight bits, but this has been modified in this version.

FUTURE DIRECTIONS

None.

SEE ALSO

sed

XBD Table 5-1 (on page 113), Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The **−C** operand is added, and the description of the **−c** operand is changed to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/31 is applied, removing text describing behavior on systems with bytes consisting of more than eight bits.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/32 is applied, updating an example in the EXAMPLES section to avoid using unspecified behavior.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/33 is applied, making a correction to the RATIONALE.

Issue 7

SD5-XCU-ERN-30 is applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Austin Group Interpretation 1003.1-2001 #132 is applied, adding rationale to the *character* construct.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0145 [325] is applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0196 [663] is applied.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

118292 **NAME**

118293 true — return true value

118294 **SYNOPSIS**

118295 `true`

118296 **DESCRIPTION**

118297 The *true* utility shall return with exit code zero.

118298 **OPTIONS**

118299 None.

118300 **OPERANDS**

118301 None.

118302 **STDIN**

118303 Not used.

118304 **INPUT FILES**

118305 None.

118306 **ENVIRONMENT VARIABLES**

118307 None.

118308 **ASYNCHRONOUS EVENTS**

118309 Default.

118310 **STDOUT**

118311 Not used.

118312 **STDERR**

118313 Not used.

118314 **OUTPUT FILES**

118315 None.

118316 **EXTENDED DESCRIPTION**

118317 None.

118318 **EXIT STATUS**

118319 Zero.

118320 **CONSEQUENCES OF ERRORS**

118321 None.

118322 **APPLICATION USAGE**

118323 This utility is typically used in shell scripts, as shown in the EXAMPLES section.

118324 Although the special built-in utility **:** (*colon*) is similar to *true*, there are some notable differences,
118325 including:

118326 • Whereas *colon* is required to accept, and do nothing with, any number of arguments, *true* is
118327 only required to accept, and discard, a first argument of `"--"`. Passing any other
118328 argument(s) to *true* may cause its behavior to differ from that described in this standard.

118329 • A non-interactive shell exits when a redirection error occurs with *colon* (unless executed via
118330 *command*), whereas with *true* it does not.

118331 • Variable assignments preceding the command name persist after executing *colon* (unless
118332 executed via *command*), but not after executing *true*.

- In shell implementations where *true* is not provided as a built-in, using *colon* avoids the overheads associated with executing an external utility.

EXAMPLES

This command is executed forever:

```
while true
do
    command
done
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.9 (on page 2499), *colon*, *command*, *false*

CHANGE HISTORY

First released in Issue 2.

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/39 is applied, replacing the terms ``None'' and ``Default'' from the STDERR and EXIT STATUS sections, respectively, with terms as defined in Section 1.4 (on page 2462).

Issue 8

Austin Group Defect 1640 is applied, clarifying the differences between *true* and **:** (*colon*).

NAME

tsort — topological sort

SYNOPSIS

```
tsort [-w] [file]
```

DESCRIPTION

The *tsort* utility shall write to standard output a totally ordered list of items consistent with a partial ordering of items contained in the input.

The application shall ensure that the input consists of pairs of items (non-empty strings) separated by one or more <blank> or <newline> characters. It is unspecified whether other white-space characters can also be used as separators. Pairs of different items shall indicate ordering. Pairs of identical items shall indicate presence, but not ordering.

If a cycle is found in the input, diagnostic or warning messages shall be written to standard error reporting that there is a cycle and indicating which nodes are in the cycle(s). If the **−w** option is specified, these messages shall be diagnostic messages. If a diagnostic message is written, the final exit status shall be non-zero.

OPTIONS

The *tsort* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

−w Set the exit status to the number of cycles found in the input, or to an implementation-defined maximum if there are more cycles than that maximum. If no cycles are found, the exit status shall be zero unless another error occurs.

OPERANDS

The following operand shall be supported:

file A pathname of a text file to order. If no *file* operand is given, the standard input shall be used.

STDIN

The standard input shall be used if no *file* operand is specified, and shall be used if the *file* operand is `'−'` and the implementation treats the `'−'` as meaning standard input. Otherwise, the standard input shall not be used. See the INPUT FILES section.

INPUT FILES

The input file shall be a text file.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *tsort*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

118399 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

118400 **ASYNCHRONOUS EVENTS**
118401 Default.

118402 **STDOUT**
118403 The standard output shall be a text file consisting of the ordered list of items, with one item per
118404 line, produced from the partially ordered input.

118405 **STDERR**
118406 The standard error shall be used only for diagnostic and warning messages.

118407 **OUTPUT FILES**
118408 None.

118409 **EXTENDED DESCRIPTION**
118410 None.

118411 **EXIT STATUS**
118412 The following exit values shall be returned:

118413 0 Successful completion.

118414 >0 An error occurred. If the **−w** option is specified and one or more cycles were found in the
118415 input, the exit status shall be the number of cycles found, or an implementation-defined
118416 maximum if more cycles than that maximum were found.

118417 **CONSEQUENCES OF ERRORS**
118418 Default.

118419 **APPLICATION USAGE**
118420 The *LC_COLLATE* variable need not affect the actions of *tsort*. The output ordering is not
118421 lexicographic, but depends on the pairs of items given as input.

118422 **EXAMPLES**
118423 The command:

```
118424 tsort <<EOF
118425 a b c c d e
118426 g g
118427 f g e f
118428 h h
118429 EOF
```

118430 produces the output:

```
118431 a
118432 b
118433 c
118434 d
118435 e
118436 f
118437 g
118438 h
```


RATIONALE

At the time that the **−w** option was added to this standard, the only known implementation reported a maximum of 255 cycles via the exit status. This has the drawback that applications cannot distinguish, from the exit status, errors caused by cycles from other errors or (when *tsort* is executed from a shell) termination by a signal. Implementations are urged to set the implementation-defined maximum number of cycles reported via the exit status to at most 124, leaving values above that maximum through 125 for other errors, and leaving values 126 and greater to have the special meanings that the shell assigns to them.

FUTURE DIRECTIONS

A future version of this standard may require that when the **−w** option is specified, the maximum number of cycles reported through the exit status of *tsort* is at most 124 and that exit status values greater than 126 are not used by *tsort*.

SEE ALSO

XBD Chapter 8 (on page 167)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #092 is applied.

The *tsort* utility is moved from the XSI option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0146 [241] is applied.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defects 1617 and 1629 are applied, clarifying how *tsort* handles cycles found in the input.

Austin Group Defect 1745 is applied, clarifying the input separator characters and the output format.

NAME

tty — return user's terminal name

SYNOPSIS

```
tty
```

DESCRIPTION

The *tty* utility shall write to the standard output the name of the terminal that is open as standard input. The name that is used shall be equivalent to the string that would be returned by the *ttyname*( ) function defined in the System Interfaces volume of POSIX.1-2024.

OPTIONS

The *tty* utility shall conform to XBD Section 12.2 (on page 215).

OPERANDS

None.

STDIN

While no input is read from standard input, standard input shall be examined to determine whether or not it is a terminal, and, if so, to determine the name of the terminal.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *tty*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If standard input is a terminal device, a pathname of the terminal as specified by the *ttyname*( ) function defined in the System Interfaces volume of POSIX.1-2024 shall be written in the following format:

```
"%s\n", <terminal name>
```

Otherwise, a message shall be written indicating that standard input is not connected to a terminal. In the POSIX locale, the *tty* utility shall use the format:

```
"not a tty\n"
```


STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Standard input is a terminal, and the output specified in STDOUT was successfully written to standard output.

1 Standard input is not a terminal, and the output specified in STDOUT was successfully written to standard output.

>1 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

This utility checks the status of the file open as standard input against that of an implementation-defined set of files. It is possible that no match can be found, or that the match found need not be the same file as that which was opened for standard input (although they are the same device).

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *isatty*( ), *ttyname*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

The SYNOPSIS is changed to indicate two forms of the command, with the second form marked as obsolete. This is a clarification and does not change the functionality published in previous issues.

Issue 6

The obsolescent **−s** option is removed.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1509 is applied, changing the EXIT STATUS section.

NAME

type — write a description of command type

SYNOPSIS

XSI

```
type name...
```

DESCRIPTION

The *type* utility shall indicate how each argument would be interpreted if used as a command name.

OPTIONS

None.

OPERANDS

The following operand shall be supported:

name A name to be interpreted.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *type*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of messages objects and message catalogs.

PATH Determine the location of *name*, as described in XBD Chapter 8 (on page 167).

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output of *type* contains information about each operand in an unspecified format. The information provided typically identifies the operand as a shell built-in, function, alias, or keyword, and where applicable, may display the operand's pathname.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

This utility is required to be intrinsic. See Section 1.7 (on page 2470) for details.

Since *type* must be aware of the contents of the current shell execution environment (such as the lists of commands, functions, and built-ins processed by *hash*), it is always provided as a shell regular built-in. If it is called in a separate utility execution environment, such as one of the following:

```
nohup type writer
find . -type f -exec type {} +
```

it might not produce accurate results.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

command, *hash*

XBD Chapter 8 (on page 167)

CHANGE HISTORY

First released in Issue 2.

Issue 8

Austin Group Defect 248 is applied, changing a command line in the APPLICATION USAGE section.

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that this utility is required to be intrinsic.

118630 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

ulimit — report or set resource limits

SYNOPSIS

```
ulimit [-H|-S] -a
```

XSI
```
ulimit [-H|-S] [-c|-d|-f|-n|-s|-t|-v] [newlimit]
```

DESCRIPTION

The *ulimit* utility shall report or set the resource limits in effect in the process in which it is executed.

Soft limits can be changed by a process to any value that is less than or equal to the hard limit. A process can (irreversibly) lower its hard limit to any value that is greater than or equal to the soft limit. Only a process with appropriate privileges can raise a hard limit.

The value **unlimited** for a resource shall be considered to be larger than any other limit value. When a resource has this limit value, the implementation shall not enforce limits on that resource. In locales other than the POSIX locale, *ulimit* may support additional non-numeric values with the same meaning as **unlimited**.

The behavior when resource limits are exceeded shall be as described in the System Interfaces volume of POSIX.1-2024 for the *setrlimit*( ) function.

OPTIONS

The *ulimit* utility shall conform to XBD Section 12.2 (on page 215), except that:

- The order in which options other than –**H**, –**S**, and –**a** are specified may be significant.
- Conforming applications shall specify each option separately; that is, grouping option letters (for example, –**fH**) need not be recognized by all implementations.

The following options shall be supported:

–H Report hard limit(s) or set only a hard limit.

–S Report soft limit(s) or set only a soft limit.

–a Report the limit value for all of the resources named below and for any implementation-specific additional resources.

–c Report, or set if the *newlimit* operand is present, the core image size limit(s) in units of 512 bytes. [RLIMIT_CORE]

–d Report, or set if the *newlimit* operand is present, the data segment size limit(s) in units of 1 024 bytes. [RLIMIT_DATA]

–f Report, or set if the *newlimit* operand is present, the file size limit(s) in units of 512 bytes. [RLIMIT_FSIZE]

–n Report, or set if the *newlimit* operand is present, the limit(s) on the number of open file descriptors, given as a number one greater than the maximum value that the system assigns to a newly-created descriptor. [RLIMIT_NOFILE]

–s Report, or set if the *newlimit* operand is present, the stack size limit(s) in units of 1 024 bytes. [RLIMIT_STACK]

XSI **–t** Report, or set if the *newlimit* operand is present, the per-process CPU time limit(s) in units of seconds. [RLIMIT_CPU]

118671 **–v** Report, or set if the *newlimit* operand is present, the address space size limit(s) in
118672 units of 1 024 bytes. [RLIMIT_AS]

118673 Where an option description is followed by [RLIMIT_*name*] it indicates which resource for the
118674 *getrlimit*( ) and *setrlimit*( ) functions, defined in the System Interfaces volume of POSIX.1-2024,
118675 the option corresponds to.

118676 If neither the **–H** nor **–S** option is specified:

118677 • If the *newlimit* operand is present, it shall be used as the new value for both the hard and
118678 soft limits.

118679 • If the *newlimit* operand is not present, **–S** shall be the default.

118680 If no options other than **–H** or **–S** are specified, the behavior shall be as if the **–f** option was
118681 (also) specified.

118682 If any option other than **–H** or **–S** is repeated, the behavior is unspecified.

118683 **OPERANDS**

118684 The following operand shall be supported:

118685 *newlimit* Either an integer value to use as the new limit(s) for the specified resource, in the
118686 units specified in OPTIONS, or a non-numeric string indicating no limit, as
118687 described in the DESCRIPTION section. Numerals in the range 0 to the maximum
118688 limit value supported by the implementation for any resource shall be syntactically
118689 recognized as numeric values.

118690 **STDIN**

118691 Not used.

118692 **INPUT FILES**

118693 None.

118694 **ENVIRONMENT VARIABLES**

118695 The following environment variables shall affect the execution of *ulimit*:

118696 *LANG* Provide a default value for the internationalization variables that are unset or null.
118697 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
118698 variables used to determine the values of locale categories.)

118699 *LC_ALL* If set to a non-empty string value, override the values of all the other
118700 internationalization variables.

118701 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
118702 characters (for example, single-byte as opposed to multi-byte characters in
118703 arguments).

118704 *LC_MESSAGES*
118705 Determine the locale that should be used to affect the format and contents of
118706 diagnostic messages written to standard error.

118707 *NLSPATH* Determine the location of messages objects and message catalogs.

118708 **ASYNCHRONOUS EVENTS**

118709 Default.

118710 **STDOUT**

118711 The standard output shall be used when no *newlimit* operand is present.

118712 If the **–a** option is specified, the output written for each resource shall consist of one line that
118713 includes:

118714 • A short phrase identifying the resource (for example ``file size'').

118715 • An indication of the units used for the resource, if the corresponding option description in
118716 OPTIONS specifies the units to be used.

118717 • The *ulimit* option used to specify the resource.

118718 • The limit value.

118719 The format used within each line is unspecified, except that the format used for the limit value
118720 shall be as described below for the case where a single limit value is written.

118721 If a single limit value is to be written; that is, the **–a** option is not specified and at most one
118722 option other than **–H** or **–S** is specified:

118723 • If the resource being reported has a numeric limit, the limit value shall be written in the
118724 following format:

118725 `"%1d\n", <limit value>`

118726 where **<***limit value***>** is the value of the limit in the units specified in OPTIONS.

118727 • If the resource being reported does not have a numeric limit, in the POSIX locale the
118728 following format shall be used:

118729 `"unlimited\n"`

118730 **STDERR**

118731 The standard error shall be used only for diagnostic messages.

118732 **OUTPUT FILES**

118733 None.

118734 **EXTENDED DESCRIPTION**

118735 None.

118736 **EXIT STATUS**

118737 The following exit values shall be returned:

118738 0 Successful completion.

118739 **>**0 A request for a higher limit was rejected or an error occurred.

118740 **CONSEQUENCES OF ERRORS**

118741 Default.

118742 **APPLICATION USAGE**

118743 This utility is required to be intrinsic. See Section 1.7 (on page 2470) for details.

118744 Since *ulimit* affects the current shell execution environment, it is always provided as a shell
118745 regular built-in. If it is called with an operand in a separate utility execution environment, such
118746 as one of the following:

```
118747 nohup ulimit -f 10000
118748 env ulimit -S -c 10000
```

118749 it does not affect the limit(s) in the caller's environment.

See also the APPLICATION USAGE for *getrlimit*( ).

EXAMPLES

Set the hard and soft file size limits to 51 200 bytes:

```
ulimit -f 100
```

Save and restore a soft resource limit (where *X* is an option letter specifying a resource):

```
saved=$(ulimit -X)
...
ulimit -X -S "$saved"
```

Execute a utility with a CPU limit of 5 minutes (using an asynchronous subshell to ensure the limit is set in a child process):

```
(ulimit -t 300; exec utility_name </dev/null) &
wait $!
```

RATIONALE

The *ulimit* utility has no equivalent of the special values RLIM_SAVED_MAX and RLIM_SAVED_CUR returned by *getrlimit*( ), as *ulimit* is required to be able to output, and accept as input, all numeric limit values supported by the system.

Implementations differ in their behavior when the **–a** option is not specified and more than one option other than **–H** or **–S** is specified. Some write output for all of the specified resources in the same format as for **–a**; others write only the value for the last specified option. Both behaviors are allowed by the standard, since the SYNOPSIS lists the options as mutually exclusive.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *getrlimit*( )

CHANGE HISTORY

First released in Issue 2.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that this utility is required to be intrinsic.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1418 is applied, adding the **–H**, **–S**, **–a**, **–c**, **–d**, **–n**, **–s**, **–t**, and **–v** options, and relating the **–f** option to the RLIMIT_FSIZE resource for *setrlimit*( ).

Austin Group Defect 1669 is applied, moving the *ulimit* utility, excluding the **–t** option, from the XSI option to the Base.

NAME

umask — get or set the file mode creation mask

SYNOPSIS

```
umask [-S] [mask]
```

DESCRIPTION

The *umask* utility shall set the file mode creation mask of the current shell execution environment (see Section 2.13, on page 2522) to the value specified by the *mask* operand. This mask shall affect the initial value of the file permission bits of subsequently created files. If *umask* is called in a subshell or separate utility execution environment, such as one of the following:

```
(umask 002)
nohup umask ...
find . -exec umask ... \;
```

it shall not affect the file mode creation mask of the caller's environment.

If the *mask* operand is not specified, the *umask* utility shall write to standard output the value of the file mode creation mask of the invoking process.

OPTIONS

The *umask* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

-S Produce symbolic output.

The default output style is unspecified, but shall be recognized on a subsequent invocation of *umask* on the same system as a *mask* operand to restore the previous file mode creation mask.

OPERANDS

The following operand shall be supported:

mask A string specifying the new file mode creation mask. The string is treated in the same way as the *mode* operand described in the EXTENDED DESCRIPTION section for *chmod*.

For a *symbolic_mode* value, the new value of the file mode creation mask shall be the logical complement of the file permission bits portion of the file mode specified by the *symbolic_mode* string.

In a *symbolic_mode* value, the permissions *op* characters `'+'` and `'-'` shall be interpreted relative to the current file mode creation mask; `'+'` shall cause the bits for the indicated permissions to be cleared in the mask; `'-'` shall cause the bits for the indicated permissions to be set in the mask.

The interpretation of *mode* values that specify file mode bits other than the file permission bits is unspecified.

In the octal integer form of *mode*, the specified bits are set in the file mode creation mask.

The file mode creation mask shall be set to the resulting numeric value.

The default output of a prior invocation of *umask* on the same system with no operand also shall be recognized as a *mask* operand.

118828 **STDIN**

118829 Not used.

118830 **INPUT FILES**

118831 None.

118832 **ENVIRONMENT VARIABLES**

118833 The following environment variables shall affect the execution of *umask*:

118834 *LANG* Provide a default value for the internationalization variables that are unset or null.
118835 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
118836 variables used to determine the values of locale categories.)

118837 *LC_ALL* If set to a non-empty string value, override the values of all the other
118838 internationalization variables.

118839 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
118840 characters (for example, single-byte as opposed to multi-byte characters in
118841 arguments).

118842 *LC_MESSAGES*
118843 Determine the locale that should be used to affect the format and contents of
118844 diagnostic messages written to standard error.

118845 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

118846 **ASYNCHRONOUS EVENTS**

118847 Default.

118848 **STDOUT**

118849 When the *mask* operand is not specified, the *umask* utility shall write a message to standard
118850 output that can later be used as a *umask mask* operand.

118851 If **–S** is specified, the message shall be in the following format:

118852 `"u=%s,g=%s,o=%s\n", <owner permissions>, <group permissions>,`
118853 `<other permissions>`

118854 where the three values shall be combinations of letters from the set {*r*, *w*, *x*}; the presence of a
118855 letter shall indicate that the corresponding bit is clear in the file mode creation mask.

118856 If a *mask* operand is specified, there shall be no output written to standard output.

118857 **STDERR**

118858 The standard error shall be used only for diagnostic messages.

118859 **OUTPUT FILES**

118860 None.

118861 **EXTENDED DESCRIPTION**

118862 None.

118863 **EXIT STATUS**

118864 The following exit values shall be returned:

118865 0 The file mode creation mask was successfully changed, or no *mask* operand was supplied.

118866 >0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

This utility is required to be intrinsic. See Section 1.7 (on page 2470) for details.

Since *umask* affects the current shell execution environment, it is generally provided as a shell regular built-in.

In contrast to the negative permission logic provided by the file mode creation mask and the octal number form of the *mask* argument, the symbolic form of the *mask* argument specifies those permissions that are left alone.

EXAMPLES

Either of the commands:

```
umask a=rx,ug+w
```

```
umask 002
```

sets the mode mask so that subsequently created files have their S_IWOTH bit cleared.

After setting the mode mask with either of the above commands, the *umask* command can be used to write out the current value of the mode mask:

```
$ umask
0002
```

(The output format is unspecified, but historical implementations use the octal integer mode format.)

```
$ umask -S
u=rwx,g=rwx,o=rx
```

Either of these outputs can be used as the mask operand to a subsequent invocation of the *umask* utility.

Assuming the mode mask is set as above, the command:

```
umask g-w
```

sets the mode mask so that subsequently created files have their S_IWGRP and S_IWOTH bits cleared.

The command:

```
umask -- -w
```

sets the mode mask so that subsequently created files have all their write bits cleared. Note that *mask* operands **−r**, **−w**, **−x** or anything beginning with a **<**hyphen-minus**>**, must be preceded by `"--"` to keep it from being interpreted as an option.

RATIONALE

Since *umask* affects the current shell execution environment, it is generally provided as a shell regular built-in. If it is called in a subshell or separate utility execution environment, such as one of the following:

```
(umask 002)
nohup umask ...
find . -exec umask ... \;
```

it does not affect the file mode creation mask of the environment of the caller.

The description of the historical utility was modified to allow it to use the symbolic modes of *chmod*. The **–s** option used in early proposals was changed to **–S** because **–s** could be confused with a *symbolic_mode* form of mask referring to the S_ISUID and S_ISGID bits.

The default output style is unspecified to permit implementors to provide migration to the new symbolic style at the time most appropriate to their users. A **–o** flag to force octal mode output was omitted because the octal mode may not be sufficient to specify all of the information that may be present in the file mode creation mask when more secure file access permission checks are implemented.

It has been suggested that trusted systems developers might appreciate ameliorating the requirement that the mode mask ``affects'' the file access permissions, since it seems access control lists might replace the mode mask to some degree. The wording has been changed to say that it affects the file permission bits, and it leaves the details of the behavior of how they affect the file access permissions to the description in the System Interfaces volume of POSIX.1-2024.

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 2 (on page 2472), *chmod*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *umask*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The octal mode is supported.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/34 is applied, making a correction to the RATIONALE.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0197 [584] is applied.

Issue 8

Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that this utility is required to be intrinsic.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

unalias — remove alias definitions

SYNOPSIS

```
unalias alias-name...
unalias -a
```

DESCRIPTION

The *unalias* utility shall remove the definition for each alias name specified. See Section 2.3.1 (on page 2477). The aliases shall be removed from the current shell execution environment; see Section 2.13 (on page 2522).

OPTIONS

The *unalias* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

−a Remove all alias definitions from the current shell execution environment.

OPERANDS

The following operand shall be supported:

alias-name The name of an alias to be removed.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *unalias*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 One of the *alias-name* operands specified did not represent a valid alias definition, or an error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

This utility is required to be intrinsic. See Section 1.7 (on page 2470) for details.

Since *unalias* affects the current shell execution environment, it is generally provided as a shell regular built-in.

EXAMPLES

None.

RATIONALE

The *unalias* description is based on that from historical KornShell implementations. Known differences exist between that and the C shell. The KornShell version was adopted to be consistent with all the other KornShell features in this volume of POSIX.1-2024, such as command line editing.

The **−a** option is the equivalent of the *unalias* * form of the C shell and is provided to address security concerns about unknown aliases entering the environment of a user (or application) through the allowable implementation-defined predefined alias route or as a result of an *ENV* file. (Although *unalias* could be used to simplify the ``secure'' shell script shown in the *command* rationale, it does not obviate the need to quote all command names. An initial call to *unalias* **−a** would have to be quoted in case there was an alias for *unalias*.)

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 2 (on page 2472), *alias*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is marked as part of the User Portability Utilities option.

Issue 7

The *unalias* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

Issue 8

Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that this utility is required to be intrinsic.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

uname — return system name

SYNOPSIS

```
uname [-amnrsv]
```

DESCRIPTION

By default, the *uname* utility shall write the operating system name to standard output. When options are specified, symbols representing one or more system characteristics shall be written to the standard output. The format and contents of the symbols are implementation-defined. On systems conforming to the System Interfaces volume of POSIX.1-2024, the symbols written shall be those supported by the *uname*( ) function as defined in the System Interfaces volume of POSIX.1-2024.

OPTIONS

The *uname* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−a Behave as though all of the options **−mnrsv** were specified.

−m Write the name of the hardware type on which the system is running to standard output.

−n Write the name of this node within an implementation-defined communications network.

−r Write the current release level of the operating system implementation.

−s Write the name of the implementation of the operating system.

−v Write the current version level of this release of the operating system implementation.

If no options are specified, the *uname* utility shall write the operating system name, as if the **−s** option had been specified.

OPERANDS

None.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *uname*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

119067 *LC_MESSAGES*
119068 Determine the locale that should be used to affect the format and contents of
119069 diagnostic messages written to standard error.

119070 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

119071 **ASYNCHRONOUS EVENTS**
119072 Default.

119073 **STDOUT**
119074 By default, the output shall be a single line of the following form:

119075 `"%s\n", <sysname>`

119076 If the –**a** option is specified, the output shall be a single line of the following form:

119077 `"%s %s %s %s %s\n", <sysname>, <nodename>, <release>,`
119078 `<version>, <machine>`

119079 Additional implementation-defined symbols may be written; all such symbols shall be written at
119080 the end of the line of output before the <newline>.

119081 If options are specified to select different combinations of the symbols, only those symbols shall
119082 be written, in the order shown above for the –**a** option. If a symbol is not selected for writing, its
119083 corresponding trailing <blank> characters also shall not be written.

119084 **STDERR**
119085 The standard error shall be used only for diagnostic messages.

119086 **OUTPUT FILES**
119087 None.

119088 **EXTENDED DESCRIPTION**
119089 None.

119090 **EXIT STATUS**
119091 The following exit values shall be returned:

119092 0 The requested information was successfully written.

119093 >0 An error occurred.

119094 **CONSEQUENCES OF ERRORS**
119095 Default.

119096 **APPLICATION USAGE**
119097 Note that any of the symbols could include embedded <space> characters, which may affect
119098 parsing algorithms if multiple options are selected for output.

119099 The node name is typically a name that the system uses to identify itself for inter-system
119100 communication addressing.

119101 **EXAMPLES**
119102 The following command:

119103 `uname -sr`

119104 writes the operating system name and release level, separated by one or more <blank>
119105 characters.

RATIONALE

It was suggested that this utility cannot be used portably since the format of the symbols is implementation-defined. The POSIX.1 working group could not achieve consensus on defining these formats in the underlying *uname*( ) function, and there was no expectation that this volume of POSIX.1-2024 would be any more successful. Some applications may still find this historical utility of value. For example, the symbols could be used for system log entries or for comparison with operator or user input.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *uname*( )

CHANGE HISTORY

First released in Issue 2.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

uncompress — expand compressed data

SYNOPSIS

XSI

```
uncompress [-cfv] [file...]
```

DESCRIPTION

Refer to *compress*.

NAME

unexpand — convert spaces to tabs

SYNOPSIS

```
unexpand [-a|-t tablist] [file...]
```

DESCRIPTION

The *unexpand* utility shall copy files or standard input to standard output, converting <blank> characters at the beginning of each line into the maximum number of <tab> characters followed by the minimum number of <space> characters needed to fill the same column positions originally filled by the translated <blank> characters. By default, tabstops shall be set at every eighth column position. Each <backspace> shall be copied to the output, and shall cause the column position count for tab calculations to be decremented; the count shall never be decremented to a value less than one.

OPTIONS

The *unexpand* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−a In addition to translating <blank> characters at the beginning of each line, translate all sequences of two or more <blank> characters immediately preceding a tab stop to the maximum number of <tab> characters followed by the minimum number of <space> characters needed to fill the same column positions originally filled by the translated <blank> characters.

−t *tablist* Specify the tab stops. The application shall ensure that the *tablist* option-argument is a single argument consisting of a single positive decimal integer or multiple positive decimal integers, separated by <blank> or <comma> characters, in ascending order. If a single number is given, tabs shall be set *tablist* column positions apart instead of the default 8. If multiple numbers are given, the tabs shall be set at those specific column positions.

The application shall ensure that each tab-stop position *N* is an integer value greater than zero, and the list shall be in strictly ascending order. This is taken to mean that, from the start of a line of output, tabbing to position *N* shall cause the next character output to be in the (*N*+1)th column position on that line. When the **−t** option is not specified, the default shall be the equivalent of specifying **−t 8** (except for the interaction with **−a**, described below).

No <space>-to-<tab> conversions shall occur for characters at positions beyond the last of those specified in a multiple tab-stop list.

When **−t** is specified, the presence or absence of the **−a** option shall be ignored; conversion shall not be limited to the processing of leading <blank> characters.

OPERANDS

The following operand shall be supported:

file A pathname of a text file to be used as input.

STDIN

See the INPUT FILES section.

INPUT FILES

The input files shall be text files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *unexpand*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the processing of <tab> and <space> characters, and for the determination of the width in column positions each character would occupy on an output device.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall be equivalent to the input files with the specified <space>-to-<tab> conversions.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

One non-intuitive aspect of *unexpand* is its restriction to leading <space> characters when neither **−a** nor **−t** is specified. Users who always want to convert all <space> characters in a file can easily alias *unexpand* to use the **−a** or **−t 8** option.

EXAMPLES

None.

RATIONALE

On several occasions, consideration was given to adding a **−t** option to the *unexpand* utility to complement the **−t** in *expand* (see *expand*). The historical intent of *unexpand* was to translate multiple <blank> characters into tab stops, where tab stops were a multiple of eight column

positions on most UNIX systems. An early proposal omitted **−t** because it seemed outside the scope of the User Portability Utilities option; it was not described in any of the base documents for IEEE Std 1003.2-1992. However, hard-coding tab stops every eight columns was not suitable for the international community and broke historical precedents for some vendors in the FORTRAN community, so **−t** was restored in conjunction with the list of valid extension categories considered by the standard developers. Thus, *unexpand* is now the logical converse of *expand*.

FUTURE DIRECTIONS

None.

SEE ALSO

expand, *tabs*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The definition of the *LC_CTYPE* environment variable is changed to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

The *unexpand* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0198 [885] is applied.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

unget — undo a previous get of an SCCS file (**DEVELOPMENT**)

SYNOPSIS

XSI

```
unget [-ns] [-r SID] file...
```

DESCRIPTION

The *unget* utility shall reverse the effect of a *get* **−e** done prior to creating the intended new delta.

OPTIONS

The *unget* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−r *SID* Uniquely identify which delta is no longer intended. (This would have been specified by *get* as the new delta.) The use of this option is necessary only if two or more outstanding *get* commands for editing on the same SCCS file were done by the same person (login name).

−s Suppress the writing to standard output of the intended delta's SID.

−n Retain the file that was obtained by *get*, which would normally be removed from the current directory.

OPERANDS

The following operands shall be supported:

file A pathname of an existing SCCS file or a directory. If *file* is a directory, the *unget* utility shall behave as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with **s.**) and unreadable files shall be silently ignored.

If exactly one *file* operand appears, and it is `'−'`, the standard input shall be read; each line of the standard input shall be taken to be the name of an SCCS file to be processed. Non-SCCS files and unreadable files shall be silently ignored.

STDIN

The standard input shall be a text file used only when the *file* operand is specified as `'−'`. Each line of the text file shall be interpreted as an SCCS pathname.

INPUT FILES

Any SCCS files processed shall be files of an unspecified format.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *unget*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall consist of a line for each file, in the following format:

```
"%s\n", <SID removed from file>
```

If there is more than one named file or if a directory or standard input is named, each pathname shall be written before each of the preceding lines:

```
"\n%s:\n", <pathname>
```

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

Any SCCS files updated shall be files of an unspecified format. During processing of a *file*, a locking *z-file*, as described in *get*, and a *q-file* (a working copy of the *p-file*), may be created and deleted. The *p-file* and *g-file*, as described in *get*, shall be deleted.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

delta, *get*, *sact*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

uniq — report or filter out repeated lines in a file

SYNOPSIS

```
uniq [-c|-d|-u] [-f fields] [-s char] [input_file [output_file]]
```

DESCRIPTION

The *uniq* utility shall read an input file comparing adjacent lines, and write one copy of each input line on the output. The second and succeeding copies of repeated adjacent input lines shall not be written. The trailing <newline> of each line in the input shall be ignored when doing comparisons.

Repeated lines in the input shall not be detected if they are not adjacent.

OPTIONS

The *uniq* utility shall conform to XBD Section 12.2 (on page 215), except that `'+'` may be recognized as an option delimiter as well as `'-'`.

The following options shall be supported:

-c Precede each output line with a count of the number of times the line occurred in the input.

-d Suppress the writing of lines that are not repeated in the input.

-f *fields* Ignore the first *fields* fields on each input line when doing comparisons, where *fields* is a positive decimal integer. A field is the maximal string matched by the basic regular expression:

```
[[:blank:]]*[^[:blank:]]*
```

If the *fields* option-argument specifies more fields than appear on an input line, a null string shall be used for comparison.

-s *chars* Ignore the first *chars* characters when doing comparisons, where *chars* shall be a positive decimal integer. If specified in conjunction with the **-f** option, the first *chars* characters after the first *fields* fields shall be ignored. If the *chars* option-argument specifies more characters than remain on an input line, a null string shall be used for comparison.

-u Suppress the writing of lines that are repeated in the input.

OPERANDS

The following operands shall be supported:

input_file A pathname of the input file. If the *input_file* operand is not specified, or if the *input_file* is `'-'`, the standard input shall be used.

output_file A pathname of the output file. If the *output_file* operand is not specified, the standard output shall be used. The results are unspecified if the file named by *output_file* is the file named by *input_file*.

STDIN

The standard input shall be used only if no *input_file* operand is specified or if *input_file* is `'-'`. See the INPUT FILES section.

INPUT FILES

The input file shall be a text file.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *uniq*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and which characters constitute a <blank> in the current locale.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall be used if no *output_file* operand is specified, and shall be used if the *output_file* operand is `'-'` and the implementation treats the `'-'` as meaning standard output. Otherwise, the standard output shall not be used. See the OUTPUT FILES section.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

If the **−c** option is specified, the output file shall be empty or each line shall be of the form:

```
"%d %s", <number of duplicates>, <line>
```

otherwise, the output file shall be empty or each line shall be of the form:

```
"%s", <line>
```

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *sort* utility can be used to cause repeated lines to be adjacent in the input file.

If the collating sequence of the current locale does not have a total ordering of all characters, the behavior of `sort | uniq` differs from `sort -u`, as `uniq` treats lines as duplicates only if they are identical, whereas `sort -u` treats lines as duplicates if they collate equally.

When using *uniq* to process pathnames, it is recommended that LC_ALL, or at least LC_CTYPE and LC_COLLATE, are set to POSIX or C in the environment, since pathnames can contain byte sequences that do not form valid characters in some locales, in which case the utility's behavior would be undefined. In the POSIX locale each byte is a valid single-byte character, and therefore this problem is avoided.

EXAMPLES

The following input file data (but flushed left) was used for a test series on *uniq*:

```
#01 foo0 bar0 foo1 bar1
#02 bar0 foo1 bar1 foo1
#03 foo0 bar0 foo1 bar1
#04
#05 foo0 bar0 foo1 bar1
#06 foo0 bar0 foo1 bar1
#07 bar0 foo1 bar1 foo0
```

What follows is a series of test invocations of the *uniq* utility that use a mixture of *uniq* options against the input file data. These tests verify the meaning of *adjacent*. The *uniq* utility views the input data as a sequence of strings delimited by `'\n'`. Accordingly, for the *fields*th member of the sequence, *uniq* interprets unique or repeated adjacent lines strictly relative to the *fields*+1th member.

1. This first example tests the line counting option, comparing each line of the input file data starting from the second field:

   ```
   uniq -c -f 1 uniq_0I.t
       1 #01 foo0 bar0 foo1 bar1
       1 #02 bar0 foo1 bar1 foo1
       1 #03 foo0 bar0 foo1 bar1
       1 #04
       2 #05 foo0 bar0 foo1 bar1
       1 #07 bar0 foo1 bar1 foo0
   ```

   The number `'2'`, prefixing the fifth line of output, signifies that the *uniq* utility detected a pair of repeated lines. Given the input data, this can only be true when *uniq* is run using the **–f 1** option (which shall cause *uniq* to ignore the first field on each input line).

2. The second example tests the option to suppress unique lines, comparing each line of the input file data starting from the second field:

   ```
   uniq -d -f 1 uniq_0I.t
   #05 foo0 bar0 foo1 bar1
   ```

3. This test suppresses repeated lines, comparing each line of the input file data starting from the second field:

   ```
   uniq -u -f 1 uniq_0I.t
   #01 foo0 bar0 foo1 bar1
   #02 bar0 foo1 bar1 foo1
   #03 foo0 bar0 foo1 bar1
   ```



```
#04
#07 bar0 foo1 bar1 foo0
```

4. This suppresses unique lines, comparing each line of the input file data starting from the third character:

```
uniq -d -s 2 uniq_0I.t
```

In the last example, the *uniq* utility found no input matching the above criteria.

RATIONALE

Some historical implementations have limited lines to be 1 080 bytes in length, which does not meet the implied {LINE_MAX} limit.

Earlier versions of this standard allowed the –*number* and +*number* options. These options are no longer specified by POSIX.1-2024 but may be present in some implementations.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

comm, *sort*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The obsolescent SYNOPSIS and associated text are removed.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/40 is applied, adding *LC_COLLATE* to the ENVIRONMENT VARIABLES section, and changing ``the application shall ensure that'' in the OUTPUT FILES section.

Issue 7

Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that `'+'` may be recognized as an option delimiter in the OPTIONS section.

Austin Group Interpretation 1003.1-2001 #092 is applied.

Austin Group Interpretation 1003.1-2001 #133 is applied, clarifying the behavior of the trailing <newline>.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

SD5-XCU-ERN-141 is applied, updating the EXAMPLES section.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0199 [963] and XCU/TC2-2008/0200 [663] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1070 is applied, changing the APPLICATION USAGE section.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

119493 Austin Group Defect 1492 is applied, changing the EXIT STATUS section.

NAME

unlink — call the *unlink*( ) function

SYNOPSIS

XSI
```
unlink file
```

DESCRIPTION

The *unlink* utility shall perform the function call:

```
unlink(file);
```

A user may need appropriate privileges to invoke the *unlink* utility.

OPTIONS

None.

OPERANDS

The following operands shall be supported:

file The pathname of an existing file.

STDIN

Not used.

INPUT FILES

Not used.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *unlink*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

None.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

link, *rm*

XBD Chapter 8 (on page 167)

XSH *unlink*( )

CHANGE HISTORY

First released in Issue 5.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

uucp — system-to-system copy

SYNOPSIS

UU

```
uucp [-cCdfjmr] [-n user] source-file... destination-file
```

DESCRIPTION

The *uucp* utility shall copy files named by the *source-file* argument to the *destination-file* argument. The files named can be on local or remote systems.

The *uucp* utility cannot guarantee support for all character encodings in all circumstances. For example, transmission data may be restricted to 7 bits by the underlying network, 8-bit data and filenames need not be portable to non-internationalized systems, and so on. Under these circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used, and that only characters defined in the portable filename character set be used for naming files. The protocol for transfer of files is unspecified by POSIX.1-2024.

Typical implementations of this utility require a communications line configured to use XBD Chapter 11 (on page 199), but other communications means may be used. On systems where there are no available communications means (either temporarily or permanently), this utility shall write an error message describing the problem and exit with a non-zero exit status.

OPTIONS

The *uucp* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−c Do not copy local file to the spool directory for transfer to the remote machine (default).

−C Force the copy of local files to the spool directory for transfer.

−d Make all necessary directories for the file copy (default).

−f Do not make intermediate directories for the file copy.

−j Write the job identification string to standard output. This job identification can be used by *uustat* to obtain the status or terminate a job.

−m Send mail to the requester when the copy is completed.

−n *user* Notify *user* on the remote system that a file was sent.

−r Do not start the file transfer; just queue the job.

OPERANDS

The following operands shall be supported:

destination-file, *source-file*

A pathname of a file to be copied to, or from, respectively. Either name can be a pathname on the local machine, or can have the form:

```
system-name!pathname
```

where *system-name* is taken from a list of system names that *uucp* knows about. The destination *system-name* can also be a list of names such as:

```
system-name!system-name!...!system-name!pathname
```

in which case, an attempt is made to send the file via the specified route to the

119598 destination. Care should be taken to ensure that intermediate nodes in the route
119599 are willing to forward information.

119600 The shell pattern matching notation characters `'?'`, `'*'`, and `"[...]"` appearing
119601 in *pathname* shall be expanded on the appropriate system.

119602 Pathnames can be one of:

119603 1. An absolute pathname.

119604 2. A pathname preceded by *~user* where *user* is a login name on the specified
119605 system and is replaced by that user's login directory. Note that if an invalid
119606 login is specified, the default is to the public directory (called *PUBDIR*; the
119607 actual location of *PUBDIR* is implementation-defined).

119608 3. A pathname preceded by *~/destination* where *destination* is appended to
119609 *PUBDIR*.

119610 **Note:** This destination is treated as a filename unless more than one file is being
119611 transferred by this request or the destination is already a directory. To
119612 ensure that it is a directory, follow the destination with a `'/'`. For
119613 example, **~/dan/** as the destination makes the directory **PUBDIR/dan** if it
119614 does not exist and puts the requested files in that directory.

119615 4. Anything else shall be prefixed by the current directory.

119616 If the result is an erroneous pathname for the remote system, the copy shall fail. If
119617 the *destination-file* is a directory, the last part of the *source-file* name shall be used.

119618 The read, write, and execute permissions given by *uucp* are implementation-
119619 defined.

119620 **STDIN**

119621 Not used.

119622 **INPUT FILES**

119623 The files to be copied are regular files.

119624 **ENVIRONMENT VARIABLES**

119625 The following environment variables shall affect the execution of *uucp*:

119626 *LANG* Provide a default value for the internationalization variables that are unset or null.
119627 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
119628 variables used to determine the values of locale categories.)

119629 *LC_ALL* If set to a non-empty string value, override the values of all the other
119630 internationalization variables.

119631 *LC_COLLATE*
119632 Determine the locale for the behavior of ranges, equivalence classes, and multi-
119633 character collating elements within bracketed filename patterns.

119634 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
119635 characters (for example, single-byte as opposed to multi-byte characters in
119636 arguments and input files) and the behavior of character classes within bracketed
119637 filename patterns (for example, `"'[[:lower:]]*'"`).

119638 *LC_MESSAGES*
119639 Determine the locale that should be used to affect the format and contents of
119640 diagnostic messages written to standard error, and informative messages written
119641 to standard output.

119642 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

119643 **ASYNCHRONOUS EVENTS**
119644 Default.

119645 **STDOUT**
119646 Not used.

119647 **STDERR**
119648 The standard error shall be used only for diagnostic messages.

119649 **OUTPUT FILES**
119650 The output files (which may be on other systems) are copies of the input files.

119651 If **−m** is used, mail files are modified.

119652 **EXTENDED DESCRIPTION**
119653 None.

119654 **EXIT STATUS**
119655 The following exit values shall be returned:

119656 0 Successful completion.

119657 >0 An error occurred.

119658 **CONSEQUENCES OF ERRORS**
119659 Default.

119660 **APPLICATION USAGE**
119661 This utility is part of the UUCP Utilities option and need not be supported by all
119662 implementations.

119663 The domain of remotely accessible files can (and for obvious security reasons usually should) be
119664 severely restricted.

119665 Note that the `'!'` character in addresses has to be escaped when using *csh* as a command
119666 interpreter because of its history substitution syntax. For *ksh* and *sh* the escape is not necessary,
119667 but may be used.

119668 As noted above, shell metacharacters appearing in pathnames are expanded on the appropriate
119669 system. On an internationalized system, this is done under the control of local settings of
119670 *LC_COLLATE* and *LC_CTYPE*. Thus, care should be taken when using bracketed filename
119671 patterns, as collation and typing rules may vary from one system to another. Also be aware that
119672 certain types of expression (that is, equivalence classes, character classes, and collating symbols)
119673 need not be supported on non-internationalized systems.

119674 **EXAMPLES**
119675 None.

119676 **RATIONALE**
119677 None.

119678 **FUTURE DIRECTIONS**
119679 If this utility is directed to create a new directory entry that contains any bytes that have the
119680 encoded value of a <newline> character, implementations are encouraged to treat this as an
119681 error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

mailx, *uuencode*, *uustat*, *uux*

XBD Chapter 8 (on page 167), Chapter 11 (on page 199), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

The UN margin codes and associated shading are removed from the **–C**, **–f**, **–j**, **–n**, and **–r** options in response to The Open Group Base Resolution bwg2001-003.

Issue 7

SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1516 is applied, adding XSI shading to text relating to *NLSPATH*.

NAME

uudecode — decode a binary file

SYNOPSIS

```
uudecode [-o outfile] [file]
```

DESCRIPTION

The *uudecode* utility shall read a file, or standard input if no file is specified, that includes data created by the *uuencode* utility. The *uudecode* utility shall scan the input file, searching for data compatible with one of the formats specified in *uuencode*, and determine the pathname for the output file from the **−o** option if given, otherwise from the input data. If the pathname for the output file is either of the magic cookies – or **/dev/stdout**, *uudecode* shall write the decoded file to standard output, otherwise it shall attempt to create or overwrite the file named by the pathname. The file access permission bits and contents for the file to be produced shall be contained in the input data. The mode bits of the created file (other than standard output) shall be set from the file access permission bits contained in the data; that is, other attributes of the mode, including the file mode creation mask (see *umask*), shall not affect the file being produced. If either of the *op* characters `'+'` and `'−'` (see *chmod*) are specified in symbolic mode, the initial mode on which those operations are based is unspecified.

If the pathname of the file resolves to an existing file and the user does not have write permission on that file, *uudecode* shall terminate with an error. If the pathname of the file resolves to an existing file and the user has write permission on that file, the existing file shall be overwritten and, if possible, the mode bits of the file (other than standard output) shall be set as described above; if the mode bits cannot be set, *uudecode* shall not treat this as an error.

If the input data was produced by *uuencode* on a system with a different number of bits per byte than on the target system, the results of *uudecode* are unspecified.

OPTIONS

The *uudecode* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported by the implementation:

−o *outfile* A pathname of a file that shall be used instead of any pathname contained in the input data. Specifying an *outfile* option-argument of – or **/dev/stdout** shall indicate standard output.

OPERANDS

The following operand shall be supported:

file The pathname of a file containing the output of *uuencode*.

STDIN

See the INPUT FILES section.

INPUT FILES

The input files shall be files containing the output of *uuencode*.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *uudecode*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If the pathname specified for the output file is – or **/dev/stdout**, the standard output shall be in the same format as the file originally encoded by *uuencode*. Otherwise, the standard output shall not be used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

The output file shall be in the same format as the file originally encoded by *uuencode*.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The user who is invoking *uudecode* must have write permission on any file being created.

The output of *uuencode* is essentially an encoded bit stream that is not cognizant of byte boundaries. It is possible that a 9-bit byte target machine can process input from an 8-bit source, if it is aware of the requirement, but the reverse is unlikely to be satisfying. Of course, the only data that is meaningful for such a transfer between architectures is generally character data.

In order to create an output file named –, it needs to be specified using an alternative pathname, for example, **–o ./–**, since – alone is considered a magic cookie by *uudecode*. Likewise, in order to write to an output file named **/dev/stdout** it also needs to be specified as, for example, **–o ///dev/stdout**.

EXAMPLES

None.

RATIONALE

Input files are not necessarily text files, as stated by an early proposal. Although the *uuencode* output is a text file, that output could have been wrapped within another file or mail message that is not a text file.

The **–o** option is not historical practice, but was added at the request of WG15 so that the user could override the target pathname without having to edit the input data itself.

In early drafts, the [–**o** *outfile*] option-argument allowed the use of – to mean standard output. The standard developers did not wish to overload the meaning of – in this manner, resulting in previous versions only using **/dev/stdout** for this purpose. POSIX.1-2024 now allows it as most implementations were already supporting – as an extension. The file **/dev/stdout** exists as a special file on most modern systems. However, the **/dev/stdout** syntax in *uudecode* does not refer to a new file. It is just a magic cookie to specify standard output.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

chmod, *umask*, *uuencode*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The –**o** *outfile* option is added, as specified in the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/35 is applied, clarifying in the DESCRIPTION that the initial mode used if either of the *op* characters is '+' or '−' is unspecified.

Issue 7

The *uudecode* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0201 [635] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1544 is applied, changing the –**o** option to require that an option-argument of – is treated as meaning standard output.

NAME

uuencode — encode a binary file

SYNOPSIS

```
uuencode [-m] [file] decode_pathname
```

DESCRIPTION

The *uuencode* utility shall write an encoded version of the named input file, or standard input if no *file* is specified, to standard output. The output shall be encoded using one of the algorithms described in the STDOUT section and shall include the file access permission bits (in *chmod* octal or symbolic notation) of the input file and the *decode_pathname*, for re-creation of the file on another system that conforms to this volume of POSIX.1-2024.

OPTIONS

The *uuencode* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported by the implementation:

−m Encode the output using the MIME Base64 algorithm described in STDOUT. If **−m** is not specified, the historical algorithm described in STDOUT shall be used.

OPERANDS

The following operands shall be supported:

decode_pathname
: The pathname of the file into which the *uudecode* utility shall place the decoded file. Specifying a *decode_pathname* operand of – or **/dev/stdout** shall indicate that *uudecode* is to use standard output. If there are characters in *decode_pathname* that are not in the portable filename character set the results are unspecified.

file A pathname of the file to be encoded.

STDIN

See the INPUT FILES section.

INPUT FILES

Input files can be files of any type.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *uuencode*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
: Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

uuencode Base64 Algorithm

The standard output shall be a text file (encoded in the character set of the current locale) that begins with the line:

`"begin-base64Δ%sΔ%s\n", <mode>, <decode_pathname>`

and ends with the line:

`"====\n"`

In both cases, the lines shall have no preceding or trailing <blank> characters.

The encoding process represents 24-bit groups of input bits as output strings of four encoded characters. Proceeding from left to right, a 24-bit input group shall be formed by concatenating three 8-bit input groups. Each 24-bit input group then shall be treated as four concatenated 6-bit groups, each of which shall be translated into a single digit in the Base64 alphabet. When encoding a bit stream via the Base64 encoding, the bit stream shall be presumed to be ordered with the most-significant bit first. That is, the first bit in the stream shall be the high-order bit in the first byte, and the eighth bit shall be the low-order bit in the first byte, and so on. Each 6-bit group is used as an index into an array of 64 printable characters, as shown in Table 3-22.

Table 3-22 uuencode Base64 Values

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

The character referenced by the index shall be placed in the output string.

The output stream (encoded bytes) shall be represented in lines of no more than 76 characters each. All line breaks or other characters not found in the table shall be ignored by decoding software (see *uudecode*).

Special processing shall be performed if fewer than 24 bits are available at the end of a message or encapsulated part of a message. A full encoding quantum shall always be completed at the

119903 end of a message. When fewer than 24 input bits are available in an input group, zero bits shall
119904 be added (on the right) to form an integral number of 6-bit groups. Output character positions
119905 that are not required to represent actual input data shall be set to the character `'='`. Since all
119906 Base64 input is an integral number of octets, only the following cases can arise:

119907 1. The final quantum of encoding input is an integral multiple of 24 bits; here, the final unit
119908 of encoded output shall be an integral multiple of 4 characters with no `'='` padding.

119909 2. The final quantum of encoding input is exactly 16 bits; here, the final unit of encoded
119910 output shall be three characters followed by one `'='` padding character.

119911 3. The final quantum of encoding input is exactly 8 bits; here, the final unit of encoded
119912 output shall be two characters followed by two `'='` padding characters.

119913 A terminating `"===="` evaluates to nothing and denotes the end of the encoded data.

119914 **uuencode Historical Algorithm**

119915 The standard output shall be a text file (encoded in the character set of the current locale) that
119916 begins with the line:

119917 `"begin␣%s␣%s\n"` *<mode>*, *<decode_pathname>*

119918 and ends with the line:

119919 `"end\n"`

119920 In both cases, the lines shall have no preceding or trailing <blank> characters.

119921 The algorithm that shall be used for lines in between **begin** and **end** takes three octets as input
119922 and writes four characters of output by splitting the input at six-bit intervals into four octets,
119923 containing data in the lower six bits only. These octets shall be converted to characters in the
119924 ISO/IEC 646:1991 standard encoded character set by adding a value of 0x20 to each octet, so
119925 that each octet is in the range [0x20,0x5f], and then optionally replacing any 0x20 octets with
119926 0x60. If necessary, these characters shall then be translated into the corresponding character
119927 codes for the codeset in use in the current locale. For example, the octet 0x41, representing `'A'`,
119928 would be translated to `'A'` in the current codeset, such as 0xc1 if it were EBCDIC; the octet 0x20,
119929 representing <space>, would optionally be replaced with 0x60, representing `'``'`, and then
119930 translated to either <space> (0x40 if EBCDIC) or `'``'` (0x79 if EBCDIC), respectively.

119931 Where the bits of two octets are combined, the least significant bits of the first octet shall be
119932 shifted left and combined with the most significant bits of the second octet shifted right. Thus
119933 the three octets *A*, *B*, *C* shall be converted into the four octets:

```
119934 0x20 + (( A >> 2                     ) & 0x3F)
119935 0x20 + (((A << 4) | ((B >> 4) & 0xF)) & 0x3F)
119936 0x20 + (((B << 2) | ((C >> 6) & 0x3)) & 0x3F)
119937 0x20 + (( C                          ) & 0x3F)
```

119938 before any replacement of 0x20 with 0x60 and translation into the local character set.

119939 Each encoded line shall contain a length character, equal to the number of characters to be
119940 decoded plus 0x20 translated to the local character set as described above, followed by between
119941 1 and 45, inclusive, encoded characters. The last encoded line, or the **begin** line if the input is
119942 empty, shall be followed by a line containing only a <space> or `'``'` character before the
119943 terminating <newline>.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The file is expanded by 35 percent (each three octets become four, plus control information) causing it to take longer to transmit.

Since this utility is intended to create files to be used for data interchange between systems with possibly different codesets, and to represent binary data as a text file, the ISO/IEC 646:1991 standard was chosen for a midpoint in the algorithm as a known reference point. The output from *uuencode* is a text file on the local system. If the output were in the ISO/IEC 646:1991 standard codeset, it might not be a text file (at least because the <newline> characters might not match), and the goal of creating a text file would be defeated. If this text file was then carried to another machine with the same codeset, it would be perfectly compatible with that system's *uudecode*. If it was transmitted over a mail system or sent to a machine with a different codeset, it is assumed that, as for every other text file, some translation mechanism would convert it (by the time it reached a user on the other system) into an appropriate codeset. This translation only makes sense from the local codeset, not if the file has been put into a ISO/IEC 646:1991 standard representation first. Similarly, files processed by *uuencode* can be placed in *pax* archives, intermixed with other text files in the same codeset.

Since *uudecode* treats a *decode_pathname* of – to mean decode to standard output, in order to specify that a file named – is to be created, *decode_pathname* should be specified using an alternative pathname, for example **./**–. Likewise, in order to specify that a file with the pathname **/dev/stdout** is to be written, *decode_pathname* should be specified as, for example, **///dev/stdout**.

EXAMPLES

None.

RATIONALE

A new algorithm was added at the request of the international community to parallel work in RFC 2045 (MIME). As with the historical *uuencode* format, the Base64 Content-Transfer-Encoding is designed to represent arbitrary sequences of octets in a form that is not humanly readable. A 65-character subset of the ISO/IEC 646:1991 standard is used, enabling 6 bits to be represented per printable character. (The extra 65th character, `'='`, is used to signify a special processing function.)

This subset has the important property that it is represented identically in all versions of the ISO/IEC 646:1991 standard, including US ASCII, and all characters in the subset are also represented identically in all versions of EBCDIC. The historical *uuencode* algorithm does not share this property, which is the reason that a second algorithm was added to the ISO POSIX-2

119990 standard.

119991 The string `"====="` was used for the termination instead of the end used in the original format
119992 because the latter is a string that could be valid encoded input.

119993 In an early draft, the **−m** option was named **−b** (for Base64), but it was renamed to reflect its
119994 relationship to the RFC 2045. A **−u** was also present to invoke the default algorithm, but since
119995 this was not historical practice, it was omitted as being unnecessary.

119996 See the RATIONALE section in *uudecode* for the derivation of the **/dev/stdout** symbol.

119997 Historically the encoding used only octets in the range [0x20,0x5f], and thus the encoded lines
119998 could contain trailing spaces, which were at risk of being stripped by whatever transport
119999 method was used to send the file. To avoid this problem some implementations use 0x60
120000 instead of 0x20, resulting in `'`'` characters instead of spaces in the output, and implementations
120001 are encouraged to do this.

120002 **FUTURE DIRECTIONS**

120003 None.

120004 **SEE ALSO**

120005 *chmod*, *mailx*, *uudecode*

120006 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

120007 **CHANGE HISTORY**

120008 First released in Issue 4.

120009 **Issue 6**

120010 This utility is marked as part of the User Portability Utilities option.

120011 The Base64 algorithm and the ability to output to **/dev/stdout** are added as specified in the
120012 IEEE P1003.2b draft standard.

120013 **Issue 7**

120014 The *uuencode* utility is moved from the User Portability Utilities option to the Base. User
120015 Portability Utilities is now an option for interactive utilities.

120016 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

120017 **Issue 8**

120018 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

120019 Austin Group Defect 1140 is applied, changing the description of the *uuencode* historical
120020 algorithm.

120021 Austin Group Defect 1544 is applied, changing the OPERANDS and APPLICATION USAGE
120022 sections.

NAME

uustat — uucp status enquiry and job control

SYNOPSIS

UU

```
uustat [-q|-k jobid|-r jobid]
uustat [-s system] [-u user]
```

DESCRIPTION

The *uustat* utility shall display the status of, or cancel, previously specified *uucp* requests, or provide general status on *uucp* connections to other systems.

When no options are given, *uustat* shall write to standard output the status of all *uucp* requests issued by the current user.

Typical implementations of this utility require a communications line configured to use XBD Chapter 11 (on page 199), but other communications means may be used. On systems where there are no available communications means (either temporarily or permanently), this utility shall write an error message describing the problem and exit with a non-zero exit status.

OPTIONS

The *uustat* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−q Write the jobs queued for each machine.

−k *jobid* Kill the *uucp* request whose job identification is *jobid*. The application shall ensure that the killed *uucp* request belongs to the person invoking *uustat* unless that user has appropriate privileges.

−r *jobid* Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup program from deleting the job until the jobs modification time reaches the limit imposed by the program.

−s *system* Write the status of all *uucp* requests for remote system *system*.

−u *user* Write the status of all *uucp* requests issued by *user*.

OPERANDS

None.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *uustat*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error, and informative messages written to standard output.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS
Default.

STDOUT
The standard output shall consist of information about each job selected, in an unspecified format. The information shall include at least the job ID, the user ID or name, and the remote system name.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
This utility is part of the UUCP Utilities option and need not be supported by all implementations.

EXAMPLES
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
uucp

XBD Chapter 8 (on page 167), Chapter 11 (on page 199), Section 12.2 (on page 215)

CHANGE HISTORY
First released in Issue 2.

Issue 6

The normative text is reworded to avoid use of the term ``must'' for application requirements.

The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

The UN margin code and associated shading are removed from the **−q** option in response to The Open Group Base Resolution bwg2001-003.

Issue 7

SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1516 is applied, adding XSI shading to text relating to *NLSPATH*.

NAME

uux — remote command execution

SYNOPSIS

UU

```
uux [-jnp] command-string
```

DESCRIPTION

The *uux* utility shall gather zero or more files from various systems, execute a shell pipeline (see Section 2.9, on page 2499) on a specified system, and then send the standard output of the command to a file on a specified system. Only the first command of a pipeline can have a *system-name*! prefix. All other commands in the pipeline shall be executed on the system of the first command.

The following restrictions are applicable to the shell pipeline processed by *uux*:

- In gathering files from different systems, pathname expansion shall not be performed by *uux*. Thus, a request such as:

  ```
  uux "c17 remsys!~/*.c"
  ```

  would attempt to copy the file named literally ***.c** to the local system.

- The redirection operators `">>"`, `"<<"`, `">|"`, and `">&"` shall not be accepted. Any use of these redirection operators shall cause this utility to write an error message describing the problem and exit with a non-zero exit status.

- The reserved word **!** cannot be used at the head of the pipeline to modify the exit status. (See the *command-string* operand description below.)

- Alias substitution shall not be performed.

A filename can be specified as for *uucp*; it can be an absolute pathname, a pathname preceded by *~name* (which is replaced by the corresponding login directory), a pathname specified as *~/dest* (*dest* is prefixed by the public directory called *PUBDIR*; the actual location of *PUBDIR* is implementation-defined), or a simple filename (which is prefixed by *uux* with the current directory). See *uucp* for the details.

The execution of commands on remote systems shall take place in an execution directory known to the *uucp* system. All files required for the execution shall be put into this directory unless they already reside on that machine. Therefore, the application shall ensure that non-local filenames (without path or machine reference) are unique within the *uux* request.

The *uux* utility shall attempt to get all files to the execution system. For files that are output files, the application shall ensure that the filename is escaped using parentheses.

The remote system shall notify the user by mail if the requested command on the remote system was disallowed or the files were not accessible. This notification can be turned off by the **–n** option.

Typical implementations of this utility require a communications line configured to use XBD Chapter 11 (on page 199), but other communications means may be used. On systems where there are no available communications means (either temporarily or permanently), this utility shall write an error message describing the problem and exit with a non-zero exit status.

The *uux* utility cannot guarantee support for all character encodings in all circumstances. For example, transmission data may be restricted to 7 bits by the underlying network, 8-bit data and filenames need not be portable to non-internationalized systems, and so on. Under these circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used

and that only characters defined in the portable filename character set be used for naming files.

OPTIONS

The *uux* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−j Write the job identification string to standard output. This job identification can be used by *uustat* to obtain the status or terminate a job.

−n Do not notify the user if the command fails.

−p Make the standard input to *uux* the standard input to the *command-string*.

OPERANDS

The following operand shall be supported:

command-string
A string made up of one or more arguments that are similar to normal command arguments, except that the command and any filenames can be prefixed by *system-name*!. A null *system-name* shall be interpreted as the local system.

STDIN

The standard input shall not be used unless the `'-'` or **−p** option is specified; in those cases, the standard input shall be made the standard input of the *command-string*.

INPUT FILES

Input files shall be selected according to the contents of *command-string*.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *uux*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall not be used unless the **−j** option is specified; in that case, the job identification string shall be written to standard output in the following format:

`"%s\n", <jobid>`

120197 **STDERR**

120198 The standard error shall be used only for diagnostic messages.

120199 **OUTPUT FILES**

120200 Output files shall be created or written, or both, according to the contents of *command-string*.

120201 If **−n** is not used, mail files shall be modified following any command or file-access failures on
120202 the remote system.

120203 **EXTENDED DESCRIPTION**

120204 None.

120205 **EXIT STATUS**

120206 The following exit values shall be returned:

120207 0 Successful completion.

120208 >0 An error occurred.

120209 **CONSEQUENCES OF ERRORS**

120210 Default.

120211 **APPLICATION USAGE**

120212 This utility is part of the UUCP Utilities option and need not be supported by all
120213 implementations.

120214 Note that, for security reasons, many installations limit the list of commands executable on
120215 behalf of an incoming request from *uux*. Many sites permit little more than the receipt of mail
120216 via *uux*.

120217 Any characters special to the command interpreter should be quoted either by quoting the entire
120218 *command-string* or quoting the special characters as individual arguments.

120219 As noted in *uucp*, shell pattern matching notation characters appearing in pathnames are
120220 expanded on the appropriate local system. This is done under the control of local settings of
120221 *LC_COLLATE* and *LC_CTYPE*. Thus, care should be taken when using bracketed filename
120222 patterns, as collation and typing rules may vary from one system to another. Also be aware that
120223 certain types of expression (that is, equivalence classes, character classes, and collating symbols)
120224 need not be supported on non-internationalized systems.

120225 **EXAMPLES**

120226 1. The following command gets **file1** from system **a** and **file2** from system **b**, executes *diff* on
120227 the local system, and puts the results in **file.diff** in the local *PUBDIR* directory. (*PUBDIR*
120228 is the *uucp* public directory on the local system.)

120229 ```
uux "!diff a!/usr/file1 b!/a4/file2 >!~/file.diff"
```

120230 2. The following command fails because *uux* places all files copied to a system in the same
120231 working directory. Although the files **xyz** are from two different systems, their filenames
120232 are the same and conflict.

120233 ```
uux "!diff a!/usr1/xyz b!/usr2/xyz >!~/xyz.diff"
```

120234 3. The following command succeeds (assuming *diff* is permitted on system **a**) because the
120235 file local to system **a** is not copied to the working directory, and hence does not conflict
120236 with the file from system **c**.

120237 ```
uux "a!diff a!/usr/xyz c!/usr/xyz >!~/xyz.diff"
```

  
```

120238 **RATIONALE**

120239 None.

120240 **FUTURE DIRECTIONS**

120241 None.

120242 **SEE ALSO**

120243 Chapter 2 (on page 2472), *uucp*, *uuencode*, *uustat*

120244 XBD Chapter 8 (on page 167), Chapter 11 (on page 199), Section 12.2 (on page 215)

120245 **CHANGE HISTORY**

120246 First released in Issue 2.

120247 **Issue 6**

120248 The obsolescent SYNOPSIS is removed.

120249 The normative text is reworded to avoid use of the term ``must'' for application requirements.

120250 The UN margin code and associated shading are removed from the **–j** option in response to The
120251 Open Group Base Resolution bwg2001-003.

120252 **Issue 7**

120253 SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.

120254 **Issue 8**

120255 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

120256 Austin Group Defect 1516 is applied, adding XSI shading to text relating to *NLSPATH*.

NAME

val — validate SCCS files (**DEVELOPMENT**)

SYNOPSIS

XSI
```
val -
val [-s] [-m name] [-r SID] [-y type] file...
```

DESCRIPTION

The *val* utility shall determine whether the specified *file* is an SCCS file meeting the characteristics specified by the options.

OPTIONS

The *val* utility shall conform to XBD Section 12.2 (on page 215), except that the usage of the `'-'` operand is not strictly as intended by the guidelines (that is, reading options and operands from standard input).

The following options shall be supported:

-m *name* Specify a *name*, which is compared with the SCCS %**M**% keyword in *file*; see *get*.

-r *SID* Specify a *SID* (SCCS Identification String), an SCCS delta number. A check shall be made to determine whether the *SID* is ambiguous (for example, **-r 1** is ambiguous because it physically does not exist but implies 1.1, 1.2, and so on, which may exist) or invalid (for example, **-r 1.0** or **-r 1.1.0** are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, a check shall be made to determine whether it actually exists.

-s Silence the diagnostic message normally written to standard output for any error that is detected while processing each named file on a given command line.

-y *type* Specify a *type*, which shall be compared with the SCCS %**Y**% keyword in *file*; see *get*.

OPERANDS

The following operands shall be supported:

file A pathname of an existing SCCS file. If exactly one *file* operand appears, and it is `'-'`, the standard input shall be read: each line shall be independently processed as if it were a command line argument list. (However, the line is not subjected to any of the shell word expansions, such as parameter expansion or quote removal.)

STDIN

The standard input shall be a text file used only when the *file* operand is specified as `'-'`.

INPUT FILES

Any SCCS files processed shall be files of an unspecified format.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *val*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

120298 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
120299 characters (for example, single-byte as opposed to multi-byte characters in
120300 arguments and input files).

120301 *LC_MESSAGES*
120302 Determine the locale that should be used to affect the format and contents of
120303 diagnostic messages written to standard error, and informative messages written
120304 to standard output.

120305 *NLSPATH* Determine the location of messages objects and message catalogs.

120306 **ASYNCHRONOUS EVENTS**
120307 Default.

120308 **STDOUT**
120309 The standard output shall consist of informative messages about either:

120310 1. Each file processed

120311 2. Each command line read from standard input

120312 If the standard input is not used, for each *file* operand yielding a discrepancy, the output line
120313 shall have the following format:

120314 `"%s: %s\n", <pathname>, <unspecified string>`

120315 If the standard input is used, for each input line yielding a discrepancy, the output shall have the
120316 following format:

120317 `"%s\n\n %s: %s\n", <input>, <pathname>, <unspecified string>`

120318 where **<***input***>** is the input line minus its terminating **<**newline**>**.

120319 **STDERR**
120320 Not used.

120321 **OUTPUT FILES**
120322 None.

120323 **EXTENDED DESCRIPTION**
120324 None.

120325 **EXIT STATUS**
120326 The 8-bit code returned by *val* shall be a disjunction of the possible errors; that is, it can be
120327 interpreted as a bit string where set bits are interpreted as follows:

120328 0x80 = Missing file argument.
120329 0x40 = Unknown or duplicate option.
120330 0x20 = Corrupted SCCS file.
120331 0x10 = Cannot open file or file not SCCS.
120332 0x08 = *SID* is invalid or ambiguous.
120333 0x04 = *SID* does not exist.
120334 0x02 = %**Y**%, **–y** mismatch.
120335 0x01 = %**M**%, **–m** mismatch.

120336 Note that *val* can process two or more files on a given command line and can process multiple
120337 command lines (when reading the standard input). In these cases an aggregate code shall be
120338 returned: a logical OR of the codes generated for each command line and file processed.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Since the *val* exit status sets the 0x80 bit, shell applications checking `"$?"` cannot tell if it terminated due to a missing file argument or receipt of a signal.

EXAMPLES

In a directory with three SCCS files—**s.x** (of **t** type ``text''), **s.y**, and **s.z** (a corrupted file)—the following command could produce the output shown:

```
val - <<EOF
-y source s.x
-m y s.y
s.z
EOF
-y source s.x
    s.x: %Y%, -y mismatch
s.z
    s.z: corrupted SCCS file
```

RATIONALE

None.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

admin, *delta*, *get*, *prs*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The Open Group Corrigendum U025/4 is applied, correcting a typographical error in the EXIT STATUS.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0147 [416] and XCU/TC1-2008/0148 [416] are applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

vi — screen-oriented (visual) display editor

SYNOPSIS

UP

```
vi [-rR] [-c command] [-t tagstring] [-w size] [file...]
```

DESCRIPTION

This utility shall be provided on systems that both support the User Portability Utilities option and define the POSIX2_CHAR_TERM symbol. On other systems it is optional.

The *vi* (visual) utility is a screen-oriented text editor. Only the open and visual modes of the editor are described in POSIX.1-2024; see the line editor *ex* for additional editing capabilities used in *vi*. The user can switch back and forth between *vi* and *ex* and execute *ex* commands from within *vi*.

This reference page uses the term *edit buffer* to describe the current working text. No specific implementation is implied by this term. All editing changes are performed on the edit buffer, and no changes to it shall affect any file until an editor command writes the file.

When using *vi*, the terminal screen acts as a window into the editing buffer. Changes made to the editing buffer shall be reflected in the screen display; the position of the cursor on the screen shall indicate the position within the editing buffer.

Certain terminals do not have all the capabilities necessary to support the complete *vi* definition. When these commands cannot be supported on such terminals, this condition shall not produce an error message such as ``not an editor command'' or report a syntax error. The implementation may either accept the commands and produce results on the screen that are the result of an unsuccessful attempt to meet the requirements of this volume of POSIX.1-2024 or report an error describing the terminal-related deficiency.

OPTIONS

The *vi* utility shall conform to XBD Section 12.2 (on page 215), except that `'+'` may be recognized as an option delimiter as well as `'-'`.

The following options shall be supported:

-c *command* See the *ex* command description of the **-c** option.

-r See the *ex* command description of the **-r** option.

-R See the *ex* command description of the **-R** option.

-t *tagstring* See the *ex* command description of the **-t** option.

-w *size* See the *ex* command description of the **-w** option.

OPERANDS

See the OPERANDS section of the *ex* command for a description of the operands supported by the *vi* command.

STDIN

If standard input is not a terminal device, the results are undefined. The standard input consists of a series of commands and input text, as described in the EXTENDED DESCRIPTION section.

If a read from the standard input returns an error, or if the editor detects an end-of-file condition from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

120421 **INPUT FILES**

120422 See the INPUT FILES section of the *ex* command for a description of the input files supported by
120423 the *vi* command.

120424 **ENVIRONMENT VARIABLES**

120425 See the ENVIRONMENT VARIABLES section of the *ex* command for the environment variables
120426 that affect the execution of the *vi* command.

120427 **ASYNCHRONOUS EVENTS**

120428 See the ASYNCHRONOUS EVENTS section of the *ex* for the asynchronous events that affect the
120429 execution of the *vi* command.

120430 **STDOUT**

120431 If standard output is not a terminal device, undefined results occur.

120432 Standard output may be used for writing prompts to the user, for informational messages, and
120433 for writing lines from the file.

120434 **STDERR**

120435 If standard output is not a terminal device, undefined results occur.

120436 The standard error shall be used only for diagnostic messages.

120437 **OUTPUT FILES**

120438 See the OUTPUT FILES section of the *ex* command for a description of the output files
120439 supported by the *vi* command.

120440 **EXTENDED DESCRIPTION**

120441 If the terminal does not have the capabilities necessary to support an unspecified portion of the
120442 *vi* definition, implementations shall start initially in *ex* mode or open mode. Otherwise, after
120443 initialization, *vi* shall be in command mode; text input mode can be entered by one of several
120444 commands used to insert or change text. In text input mode, <ESC> can be used to return to
120445 command mode; other uses of <ESC> are described later in this section; see Terminate
120446 Command or Input Mode (on page 3536).

120447 **Initialization in ex and vi**

120448 See Initialization in ex and vi (on page 2842) for a description of *ex* and *vi* initialization for the *vi*
120449 utility.

120450 **Command Descriptions in vi**

120451 The following symbols are used in this reference page to represent arguments to commands.

120452 *buffer* See the description of *buffer* in the EXTENDED DESCRIPTION section of the *ex* utility;
120453 see Command Descriptions in ex (on page 2852).

120454 In open and visual mode, when a command synopsis shows both [*buffer*] and [*count*]
120455 preceding the command name, they can be specified in either order.

120456 *count* A positive integer used as an optional argument to most commands, either to give a
120457 repeat count or as a size. This argument is optional and shall default to 1 unless
120458 otherwise specified.

120459 The Synopsis lines for the *vi* commands <control>-G, <control>-L, <control>-R,
120460 <control>-], **%**, **&**, **^**, **D**, **m**, **M**, **Q**, **u**, **U**, and **ZZ** do not have *count* as an optional
120461 argument. Regardless, it shall not be an error to specify a *count* to these commands, and
120462 any specified *count* shall be ignored.

120463 *motion* An optional trailing argument used by the **!**, **<**, **>**, **c**, **d**, and **y** commands, which is used
120464 to indicate the region of text that shall be affected by the command. The motion can be
120465 either one of the command characters repeated or one of several other *vi* commands
120466 (listed in the following table). Each of the applicable commands specifies the region of
120467 text matched by repeating the command; each command that can be used as a motion
120468 command specifies the region of text it affects.

120469 Commands that take *motion* arguments operate on either lines or characters, depending
120470 on the circumstances. When operating on lines, all lines that fall partially or wholly
120471 within the text region specified for the command shall be affected. When operating on
120472 characters, only the exact characters in the specified text region shall be affected. Each
120473 motion command specifies this individually.

120474 When commands that may be motion commands are not used as motion commands,
120475 they shall set the current position to the current line and column as specified.

120476 The following commands shall be valid cursor motion commands:

120477 `<apostrophe>` `(` `-` `j` `H`
120478 `<carriage-return>` `)` `$` `k` `L`
120479 `<comma>` `[[` `%` `l` `M`
120480 `<control>-H` `]]` `_` `n` `N`
120481 `<control>-N` `{` `;` `t` `T`
120482 `<control>-P` `}` `?` `w` `W`
120483 `<grave-accent>` `^` `b` `B`
120484 `<newline>` `+` `e` `E`
120485 `<space>` `|` `f` `F`
120486 `<zero>` `/` `h` `G`

120487 Any *count* that is specified to a command that has an associated motion command shall
120488 be applied to the motion command. If a *count* is applied to both the command and its
120489 associated motion command, the effect shall be multiplicative.

120490 The following symbols are used in this section to specify locations in the edit buffer:

120491 *current character*
120492 The character that is currently indicated by the cursor.

120493 *end of a line*
120494 The point located between the last non-<newline> (if any) and the terminating
120495 <newline> of a line. For an empty line, this location coincides with the beginning of the
120496 line.

120497 *end of the edit buffer*
120498 The location corresponding to the end of the last line in the edit buffer.

120499 The following symbols are used in this section to specify command actions:

120500 *bigword* In the POSIX locale, *vi* shall recognize four kinds of *bigwords*:

120501 1. A maximal sequence of non-<blank> characters preceded and followed by
120502 <blank> characters or the beginning or end of a line or the edit buffer

120503 2. One or more sequential blank lines

120504 3. The first character in the edit buffer

4. The last non-<newline> in the edit buffer

word In the POSIX locale, *vi* shall recognize five kinds of words:

1. A maximal sequence of letters, digits, and underscores, delimited at both ends by:
   - — Characters other than letters, digits, or underscores
   - — The beginning or end of a line
   - — The beginning or end of the edit buffer
2. A maximal sequence of characters other than letters, digits, underscores, or <blank> characters, delimited at both ends by:
   - — A letter, digit, underscore
   - — <blank> characters
   - — The beginning or end of a line
   - — The beginning or end of the edit buffer
3. One or more sequential blank lines
4. The first character in the edit buffer
5. The last non-<newline> in the edit buffer

section boundary

A *section boundary* is one of the following:

1. A line whose first character is a <form-feed>
2. A line whose first character is an open curly brace (`'{'`)
3. A line whose first character is a <period> and whose second and third characters match a two-character pair in the **sections** edit option (see *ex*)
4. A line whose first character is a <period> and whose only other character matches the first character of a two-character pair in the **sections** edit option, where the second character of the two-character pair is a <space>
5. The first line of the edit buffer
6. The last line of the edit buffer if the last line of the edit buffer is empty or if it is a]] or } command; otherwise, the last non-<newline> of the last line of the edit buffer

paragraph boundary

A *paragraph boundary* is one of the following:

1. A section boundary
2. A line whose first character is a <period> and whose second and third characters match a two-character pair in the **paragraphs** edit option (see *ex*)
3. A line whose first character is a <period> and whose only other character matches the first character of a two-character pair in the *paragraphs* edit option, where the second character of the two-character pair is a <space>

4. One or more sequential blank lines

remembered search direction
See the description of *remembered search direction* in *ex*.

sentence boundary
A *sentence boundary* is one of the following:

1. A paragraph boundary
2. The first non-<blank> that occurs after a paragraph boundary
3. The first non-<blank> that occurs after a <period> (`'.'`), <exclamation-mark> (`'!'`), or <question-mark> (`'?'`), followed by two <space> characters or the end of a line; any number of closing parenthesis (`')'`), closing brackets (`']'`), double-quote (`'"'`), or single-quote (<apostrophe>) characters can appear between the punctuation mark and the two <space> characters or end-of-line

In the remainder of the description of the *vi* utility, the term ``buffer line'' refers to a line in the edit buffer and the term ``display line'' refers to the line or lines on the display screen used to display one buffer line. The term ``current line'' refers to a specific ``buffer line''.

If there are display lines on the screen for which there are no corresponding buffer lines because they correspond to lines that would be after the end of the file, they shall be displayed as a single <tilde> (`'~'`) character, plus the terminating <newline>.

The last line of the screen shall be used to report errors or display informational messages. It shall also be used to display the input for ``line-oriented commands'' (**/**, **?**, **:**, and **!**). When a line-oriented command is executed, the editor shall enter text input mode on the last line on the screen, using the respective command characters as prompt characters. (In the case of the **!** command, the associated motion shall be entered by the user before the editor enters text input mode.) The line entered by the user shall be terminated by a <newline>, a non-<control>-V-escaped <carriage-return>, or unescaped <ESC>. It is unspecified if more characters than require a display width minus one column number of screen columns can be entered.

If any command is executed that overwrites a portion of the screen other than the last line of the screen (for example, the *ex* **suspend** or **!** commands), other than the *ex* **shell** command, the user shall be prompted for a character before the screen is refreshed and the edit session continued.

<tab> characters shall take up the number of columns on the screen set by the **tabstop** edit option (see *ex*), unless there are less than that number of columns before the display margin that will cause the displayed line to be folded; in this case, they shall only take up the number of columns up to that boundary.

The cursor shall be placed on the current line and relative to the current column as specified by each command described in the following sections.

In open mode, if the current line is not already displayed, then it shall be displayed.

In visual mode, if the current line is not displayed, then the lines that are displayed shall be expanded, scrolled, or redrawn to cause an unspecified portion of the current line to be displayed. If the screen is redrawn, no more than the number of display lines specified by the value of the **window** edit option shall be displayed (unless the current line cannot be completely displayed in the number of display lines specified by the **window** edit option) and the current line shall be positioned as close to the center of the displayed lines as possible (within the constraints imposed by the distance of the line from the beginning or end of the edit buffer). If the current line is before the first line in the display and the screen is scrolled, an unspecified

120587 portion of the current line shall be placed on the first line of the display. If the current line is after
120588 the last line in the display and the screen is scrolled, an unspecified portion of the current line
120589 shall be placed on the last line of the display.

120590 In visual mode, if a line from the edit buffer (other than the current line) does not entirely fit into
120591 the lines at the bottom of the display that are available for its presentation, the editor may choose
120592 not to display any portion of the line. The lines of the display that do not contain text from the
120593 edit buffer for this reason shall each consist of a single `'@'` character.

120594 In visual mode, the editor may choose for unspecified reasons to not update lines in the display
120595 to correspond to the underlying edit buffer text. The lines of the display that do not correctly
120596 correspond to text from the edit buffer for this reason shall consist of a single `'@'` character (plus
120597 the terminating <newline>), and the <control>-R command shall cause the editor to update the
120598 screen to correctly represent the edit buffer.

120599 Open and visual mode commands that set the current column set it to a column position in the
120600 display, and not a character position in the line. In this case, however, the column position in the
120601 display shall be calculated for an infinite width display; for example, the column related to a
120602 character that is part of a line that has been folded onto additional screen lines is offset from the
120603 display line column where the buffer line begins, not from the beginning of a particular display
120604 line.

120605 The display cursor column in the display is based on the value of the current column, as follows,
120606 with each rule applied in turn:

120607 1. If the current column is after the last display line column used by the displayed line, the
120608 display cursor column shall be set to the last display line column occupied by the last
120609 non-<newline> in the current line; otherwise, the display cursor column shall be set to the
120610 current column.

120611 2. If the character of which some portion is displayed in the display line column specified by
120612 the display cursor column requires more than a single display line column:

120613 a. If in text input mode, the display cursor column shall be adjusted to the first
120614 display line column in which any portion of that character is displayed.

120615 b. Otherwise, the display cursor column shall be adjusted to the last display line
120616 column in which any portion of that character is displayed.

120617 The current column shall not be changed by these adjustments to the display cursor column.

120618 If an error occurs during the parsing or execution of a *vi* command:

120619 • The terminal shall be alerted. Execution of the *vi* command shall stop, and the cursor (for
120620 example, the current line and column) shall not be further modified.

120621 • Unless otherwise specified by the following command sections, it is unspecified whether
120622 an informational message shall be displayed.

120623 • Any partially entered *vi* command shall be discarded.

120624 • If the *vi* command resulted from a **map** expansion, all characters from that **map** expansion
120625 shall be discarded, except as otherwise specified by the **map** command (see *ex*).

120626 • If the *vi* command resulted from the execution of a buffer, no further commands caused by
120627 the execution of the buffer shall be executed.

120628 **Page Backwards**

120629 *Synopsis*: **[***count***]** <control>-B

120630 If in open mode, the <control>-B command shall behave identically to the **z** command.
120631 Otherwise, if the current line is the first line of the edit buffer, it shall be an error.

120632 If the **window** edit option is less than 3, display a screen where the last line of the display shall
120633 be some portion of:

120634 (*current first line*) −1

120635 otherwise, display a screen where the first line of the display shall be some portion of:

120636 (*current first line*) − *count* x ((window edit option) −2)

120637 If this calculation would result in a line that is before the first line of the edit buffer, the first line
120638 of the display shall display some portion of the first line of the edit buffer.

120639 *Current line*: If no lines from the previous display remain on the screen, set to the last line of the
120640 display; otherwise, set to (*line* – the number of new lines displayed on this screen).

120641 *Current column*: Set to non-<blank>.

120642 **Scroll Forward**

120643 *Synopsis*: **[***count***]** <control>-D

120644 If the current line is the last line of the edit buffer, it shall be an error.

120645 If no *count* is specified, *count* shall default to the *count* associated with the previous <control>-D
120646 or <control>-U command. If there was no previous <control>-D or <control>-U command, *count*
120647 shall default to the value of the **scroll** edit option.

120648 If in open mode, write lines starting with the line after the current line, until *count* lines or the
120649 last line of the file have been written.

120650 *Current line*: If the current line + *count* is past the last line of the edit buffer, set to the last line of
120651 the edit buffer; otherwise, set to the current line + *count*.

120652 *Current column*: Set to non-<blank>.

120653 **Scroll Forward by Line**

120654 *Synopsis*: **[***count***]** <control>-E

120655 Display the line count lines after the last line currently displayed.

120656 If the last line of the edit buffer is displayed, it shall be an error. If there is no line *count* lines
120657 after the last line currently displayed, the last line of the display shall display some portion of
120658 the last line of the edit buffer.

120659 *Current line*: Unchanged if the previous current character is displayed; otherwise, set to the first
120660 line displayed.

120661 *Current column*: Unchanged.

120662 **Page Forward**

120663 *Synopsis*: **[***count***]** <control>-F

120664 If in open mode, the **<**control**>**-F command shall behave identically to the **z** command.
120665 Otherwise, if the current line is the last line of the edit buffer, it shall be an error.

120666 If the **window** edit option is less than 3, display a screen where the first line of the display shall
120667 be some portion of:

120668 (*current last line*) +1

120669 otherwise, display a screen where the first line of the display shall be some portion of:

120670 (*current first line*) + *count* x ((window edit option) −2)

120671 If this calculation would result in a line that is after the last line of the edit buffer, the last line of
120672 the display shall display some portion of the last line of the edit buffer.

120673 *Current line*: If no lines from the previous display remain on the screen, set to the first line of the
120674 display; otherwise, set to (*line* + the number of new lines displayed on this screen).

120675 *Current column*: Set to non-**<**blank**>**.

120676 **Display Information**

120677 *Synopsis*: <control>-G

120678 This command shall be equivalent to the *ex* **file** command.

120679 **Move Cursor Backwards**

120680 *Synopsis*: **[***count***]** <control>-H
120681 **[***count***]** h
120682 the current *erase* character (see stty)

120683 If there are no characters before the current character on the current line, it shall be an error. If
120684 there are less than *count* previous characters on the current line, *count* shall be adjusted to the
120685 number of previous characters on the line.

120686 If used as a motion command:

120687 1. The text region shall be from the character before the starting cursor up to and including
120688 the *count*th character before the starting cursor.

120689 2. Any text copied to a buffer shall be in character mode.

120690 If not used as a motion command:

120691 *Current line*: Unchanged.

120692 *Current column*: Set to (*column* − the number of columns occupied by *count* characters ending
120693 with the previous current column).

120694 **Move Down**

120695 *Synopsis*: **[***count***]** <newline>
120696 **[***count***]** <control>-J
120697 **[***count***]** <control>-M
120698 **[***count***]** <control>-N
120699 **[***count***]** j
120700 **[***count***]** <carriage-return>
120701 **[***count***]** +

120702 If there are less than *count* lines after the current line in the edit buffer, it shall be an error.

120703 If used as a motion command:

120704 1. The text region shall include the starting line and the next *count* – 1 lines.

120705 2. Any text copied to a buffer shall be in line mode.

120706 If not used as a motion command:

120707 *Current line*: Set to *current line*+ *count*.

120708 *Current column*: Set to non-<blank> for the <carriage-return>, <control>-M, and **+** commands;
120709 otherwise, unchanged.

120710 **Clear and Redisplay**

120711 *Synopsis*: <control>-L

120712 If in open mode, clear the screen and redisplay the current line. Otherwise, clear and redisplay
120713 the screen.

120714 *Current line*: Unchanged.

120715 *Current column*: Unchanged.

120716 **Move Up**

120717 *Synopsis*: **[***count***]** <control>-P
120718 **[***count***]** k
120719 **[***count***]** -

120720 If there are less than *count* lines before the current line in the edit buffer, it shall be an error.

120721 If used as a motion command:

120722 1. The text region shall include the starting line and the previous *count* lines.

120723 2. Any text copied to a buffer shall be in line mode.

120724 If not used as a motion command:

120725 *Current line*: Set to *current line* – *count*.

120726 *Current column*: Set to non-<blank> for the – command; otherwise, unchanged.

120727 **Redraw Screen**

120728 *Synopsis*: `<control>-R`

120729 If any lines have been deleted from the display screen and flagged as deleted on the terminal
120730 using the @ convention (see the beginning of the EXTENDED DESCRIPTION section), they shall
120731 be redisplayed to match the contents of the edit buffer.

120732 It is unspecified whether lines flagged with @ because they do not fit on the terminal display
120733 shall be affected.

120734 *Current line*: Unchanged.

120735 *Current column*: Unchanged.

120736 **Scroll Backward**

120737 *Synopsis*: **[***count***]** `<control>-U`

120738 If the current line is the first line of the edit buffer, it shall be an error.

120739 If no *count* is specified, *count* shall default to the *count* associated with the previous <control>-D
120740 or <control>-U command. If there was no previous <control>-D or <control>-U command, *count*
120741 shall default to the value of the **scroll** edit option.

120742 *Current line*: If *count* is greater than the current line, set to 1; otherwise, set to the current line –
120743 *count*.

120744 *Current column*: Set to non-<blank>.

120745 **Scroll Backward by Line**

120746 *Synopsis*: **[***count***]** `<control>-Y`

120747 Display the line *count* lines before the first line currently displayed.

120748 If the current line is the first line of the edit buffer, it shall be an error. If this calculation would
120749 result in a line that is before the first line of the edit buffer, the first line of the display shall
120750 display some portion of the first line of the edit buffer.

120751 *Current line*: Unchanged if the previous current character is displayed; otherwise, set to the first
120752 line displayed.

120753 *Current column*: Unchanged.

120754 **Edit the Alternate File**

120755 *Synopsis*: `<control>-^`

120756 This command shall be equivalent to the *ex* **edit** command, with the alternate pathname as its
120757 argument.

120758 **Terminate Command or Input Mode**

120759 *Synopsis*: `<ESC>`

120760 If a partial *vi* command (as defined by at least one, non-*count* character) has been entered,
120761 discard the *count* and the command character(s).

120762 Otherwise, if no command characters have been entered, and the <ESC> was the result of a map
120763 expansion, the terminal shall be alerted and the <ESC> character shall be discarded, but it shall
120764 not be an error.

120765 Otherwise, it shall be an error.

120766 *Current line*: Unchanged.

120767 *Current column*: Unchanged.

120768 **Search for tagstring**

120769 *Synopsis*: `<control>-]`

120770 If the current character is not a word or <blank>, it shall be an error.

120771 This command shall be equivalent to the *ex* **tag** command, with the argument to that command
120772 defined as follows.

120773 If the current character is a <blank>:

120774 1. Skip all <blank> characters after the cursor up to the end of the line.

120775 2. If the end of the line is reached, it shall be an error.

120776 Then, the argument to the *ex* **tag** command shall be the current character and all subsequent
120777 characters, up to the first non-word character or the end of the line.

120778 **Move Cursor Forward**

120779 *Synopsis*: **[***count***]** `<space>`
120780 **[***count***]** `l` (ell)

120781 If there are less than *count* non-<newline> characters after the cursor on the current line, *count*
120782 shall be adjusted to the number of non-<newline> characters after the cursor on the line.

120783 If used as a motion command:

120784 1. If the current or *count*th character after the cursor is the last non-<newline> in the line, the
120785 text region shall be comprised of the current character up to and including the last
120786 non-<newline> in the line. Otherwise, the text region shall be from the current character
120787 up to, but not including, the *count*th character after the cursor.

120788 2. Any text copied to a buffer shall be in character mode.

120789 If not used as a motion command:

120790 If there are no non-<newline> characters after the current character on the current line, it shall be
120791 an error.

120792 *Current line*: Unchanged.

120793 *Current column*: Set to the last column that displays any portion of the *count*th character after the
120794 current character.

120795 **Replace Text with Results from Shell Command**

120796 *Synopsis*: **[***count***]** **!** *motion shell-commands* <newline>

120797 If the motion command is the **!** command repeated:

120798 1. If the edit buffer is empty and no *count* was supplied, the command shall be the
120799 equivalent of the *ex* **:read !** command, with the text input, and no text shall be copied to
120800 any buffer.

120801 2. Otherwise:

120802 a. If there are less than *count* –1 lines after the current line in the edit buffer, it shall be
120803 an error.

120804 b. The text region shall be from the current line up to and including the next *count* –1
120805 lines.

120806 Otherwise, the text region shall be the lines in which any character of the text region specified by
120807 the motion command appear.

120808 Any text copied to a buffer shall be in line mode.

120809 This command shall be equivalent to the *ex* **!** command for the specified lines.

120810 **Move Cursor to End-of-Line**

120811 *Synopsis*: **[***count***]** **$**

120812 It shall be an error if there are less than (*count* –1) lines after the current line in the edit buffer.

120813 If used as a motion command:

120814 1. If *count* is 1:

120815 a. It shall be an error if the line is empty.

120816 b. Otherwise, the text region shall consist of all characters from the starting cursor to
120817 the last non-<newline> in the line, inclusive, and any text copied to a buffer shall
120818 be in character mode.

120819 2. Otherwise, if the starting cursor position is at or before the first non-<blank> in the line,
120820 the text region shall consist of the current and the next *count* –1 lines, and any text saved
120821 to a buffer shall be in line mode.

120822 3. Otherwise, the text region shall consist of all characters from the starting cursor to the last
120823 non-<newline> in the line that is *count* –1 lines forward from the current line, and any text
120824 copied to a buffer shall be in character mode.

120825 If not used as a motion command:

120826 *Current line*: Set to the *current line* + *count*–1.

120827 *Current column*: The current column is set to the last display line column of the last
120828 non-<newline> in the line, or column position 1 if the line is empty.

120829 The current column shall be adjusted to be on the last display line column of the last
120830 non-<newline> of the current line as subsequent commands change the current line, until a
120831 command changes the current column.

120832 **Move to Matching Character**

120833 *Synopsis*: %

120834 If the character at the current position is not a parenthesis, bracket, or curly brace, search
120835 forward in the line to the first one of those characters. If no such character is found, it shall be an
120836 error.

120837 The matching character shall be the parenthesis, bracket, or curly brace matching the
120838 parenthesis, bracket, or curly brace, respectively, that was at the current position or that was
120839 found on the current line.

120840 Matching shall be determined as follows, for an open parenthesis:

120841 1. Set a counter to 1.
120842 2. Search forwards until a parenthesis is found or the end of the edit buffer is reached.
120843 3. If the end of the edit buffer is reached, it shall be an error.
120844 4. If an open parenthesis is found, increment the counter by 1.
120845 5. If a close parenthesis is found, decrement the counter by 1.
120846 6. If the counter is zero, the current character is the matching character.

120847 Matching for a close parenthesis shall be equivalent, except that the search shall be backwards,
120848 from the starting character to the beginning of the buffer, a close parenthesis shall increment the
120849 counter by 1, and an open parenthesis shall decrement the counter by 1.

120850 Matching for brackets and curly braces shall be equivalent, except that searching shall be done
120851 for open and close brackets or open and close curly braces. It is implementation-defined whether
120852 other characters are searched for and matched as well.

120853 If used as a motion command:

120854 1. If the matching cursor was after the starting cursor in the edit buffer, and the starting
120855 cursor position was at or before the first non-<blank> non-<newline> in the starting line,
120856 and the matching cursor position was at or after the last non-<blank> non-<newline> in
120857 the matching line, the text region shall consist of the current line to the matching line,
120858 inclusive, and any text copied to a buffer shall be in line mode.
120859 2. If the matching cursor was before the starting cursor in the edit buffer, and the starting
120860 cursor position was at or after the last non-<blank> non-<newline> in the starting line,
120861 and the matching cursor position was at or before the first non-<blank> non-<newline> in
120862 the matching line, the text region shall consist of the current line to the matching line,
120863 inclusive, and any text copied to a buffer shall be in line mode.
120864 3. Otherwise, the text region shall consist of the starting character to the matching character,
120865 inclusive, and any text copied to a buffer shall be in character mode.

120866 If not used as a motion command:

120867 *Current line*: Set to the line where the matching character is located.

120868 *Current column*: Set to the last column where any portion of the matching character is displayed.

120869 **Repeat Substitution**

120870 *Synopsis*: `&`

120871 Repeat the previous substitution command. This command shall be equivalent to the *ex* **&**
120872 command with the current line as its addresses, and without *options*, *count*, or *flags*.

120873 **Return to Previous Context at Beginning of Line**

120874 *Synopsis*: `' character`

120875 It shall be an error if there is no line in the edit buffer marked by *character*.

120876 If used as a motion command:

120877 1. If the starting cursor is after the marked cursor, then the locations of the starting cursor
120878 and the marked cursor in the edit buffer shall be logically swapped.

120879 2. The text region shall consist of the starting line up to and including the marked line, and
120880 any text copied to a buffer shall be in line mode.

120881 If not used as a motion command:

120882 *Current line*: Set to the line referenced by the mark.

120883 *Current column*: Set to non-<blank>.

120884 **Return to Previous Context**

120885 *Synopsis*: ``` ` character```

120886 It shall be an error if the marked line is no longer in the edit buffer. If the marked line no longer
120887 contains a character in the saved numbered character position, it shall be as if the marked
120888 position is the first non-<blank>.

120889 If used as a motion command:

120890 1. It shall be an error if the marked cursor references the same character in the edit buffer as
120891 the starting cursor.

120892 2. If the starting cursor is after the marked cursor, then the locations of the starting cursor
120893 and the marked cursor in the edit buffer shall be logically swapped.

120894 3. If the starting line is empty or the starting cursor is at or before the first non-<blank>
120895 non-<newline> of the starting line, and the marked cursor line is empty or the marked
120896 cursor references the first character of the marked cursor line, the text region shall consist
120897 of all lines containing characters from the starting cursor to the line before the marked
120898 cursor line, inclusive, and any text copied to a buffer shall be in line mode.

120899 4. Otherwise, if the marked cursor line is empty or the marked cursor references a character
120900 at or before the first non-<blank> non-<newline> of the marked cursor line, the region of
120901 text shall be from the starting cursor to the last non-<newline> of the line before the
120902 marked cursor line, inclusive, and any text copied to a buffer shall be in character mode.

120903 5. Otherwise, the region of text shall be from the starting cursor (inclusive), to the marked
120904 cursor (exclusive), and any text copied to a buffer shall be in character mode.

120905 If not used as a motion command:

120906 *Current line*: Set to the line referenced by the mark.

120907 *Current column*: Set to the last column in which any portion of the character referenced by the

120908 mark is displayed.

120909 **Return to Previous Section**

120910 *Synopsis*: **[***count***]** [[

120911 Move the cursor backward through the edit buffer to the first character of the previous section
120912 boundary, *count* times.

120913 If used as a motion command:

120914 1. If the starting cursor was at the first character of the starting line or the starting line was
120915 empty, and the first character of the boundary was the first character of the boundary line,
120916 the text region shall consist of the current line up to and including the line where the
120917 *count*th next boundary starts, and any text copied to a buffer shall be in line mode.

120918 2. If the boundary was the last line of the edit buffer or the last non-<newline> of the last
120919 line of the edit buffer, the text region shall consist of the last character in the edit buffer up
120920 to and including the starting character, and any text saved to a buffer shall be in character
120921 mode.

120922 3. Otherwise, the text region shall consist of the starting character up to but not including
120923 the first character in the *count*th next boundary, and any text copied to a buffer shall be in
120924 character mode.

120925 If not used as a motion command:

120926 *Current line*: Set to the line where the *count*th next boundary in the edit buffer starts.

120927 *Current column*: Set to the last column in which any portion of the first character of the *count*th
120928 next boundary is displayed, or column position 1 if the line is empty.

120929 **Move to Next Section**

120930 *Synopsis*: **[***count***]**]]

120931 Move the cursor forward through the edit buffer to the first character of the next section
120932 boundary, *count* times.

120933 If used as a motion command:

120934 1. If the starting cursor was at the first character of the starting line or the starting line was
120935 empty, and the first character of the boundary was the first character of the boundary line,
120936 the text region shall consist of the current line up to and including the line where the
120937 *count*th previous boundary starts, and any text copied to a buffer shall be in line mode.

120938 2. If the boundary was the first line of the edit buffer, the text region shall consist of the first
120939 character in the edit buffer up to but not including the starting character, and any text
120940 copied to a buffer shall be in character mode.

120941 3. Otherwise, the text region shall consist of the first character in the *count*th previous
120942 section boundary up to but not including the starting character, and any text copied to a
120943 buffer shall be in character mode.

120944 If not used as a motion command:

120945 *Current line*: Set to the line where the *count*th previous boundary in the edit buffer starts.

120946 *Current column*: Set to the last column in which any portion of the first character of the *count*th
120947 previous boundary is displayed, or column position 1 if the line is empty.

120948 **Move to First Non-<blank> Position on Current Line**

120949 *Synopsis*: ^

120950 If used as a motion command:

120951 1. If the line has no non-<blank> non-<newline> characters, or if the cursor is at the first
120952 non-<blank> non-<newline> of the line, it shall be an error.

120953 2. If the cursor is before the first non-<blank> non-<newline> of the line, the text region
120954 shall be comprised of the current character, up to, but not including, the first non-<blank>
120955 non-<newline> of the line.

120956 3. If the cursor is after the first non-<blank> non-<newline> of the line, the text region shall
120957 be from the character before the starting cursor up to and including the first non-<blank>
120958 non-<newline> of the line.

120959 4. Any text copied to a buffer shall be in character mode.

120960 If not used as a motion command:

120961 *Current line*: Unchanged.

120962 *Current column*: Set to non-<blank>.

120963 **Current and Line Above**

120964 *Synopsis*: **[***count***]** _

120965 If there are less than *count* –1 lines after the current line in the edit buffer, it shall be an error.

120966 If used as a motion command:

120967 1. If *count* is less than 2, the text region shall be the current line.

120968 2. Otherwise, the text region shall include the starting line and the next *count* –1 lines.

120969 3. Any text copied to a buffer shall be in line mode.

120970 If not used as a motion command:

120971 *Current line*: Set to current line + *count* –1.

120972 *Current column*: Set to non-<blank>.

120973 **Move Back to Beginning of Sentence**

120974 *Synopsis*: **[***count***]** (

120975 Move backward to the beginning of a sentence. This command shall be equivalent to the **[[**
120976 command, with the exception that sentence boundaries shall be used instead of section
120977 boundaries.

120978 **Move Forward to Beginning of Sentence**

120979 *Synopsis*: **[***count***]**)

120980 Move forward to the beginning of a sentence. This command shall be equivalent to the **]]**
120981 command, with the exception that sentence boundaries shall be used instead of section
120982 boundaries.

120983 **Move Back to Preceding Paragraph**

120984 *Synopsis*: **[***count***]** {

120985 Move back to the beginning of the preceding paragraph. This command shall be equivalent to
120986 the **[[** command, with the exception that paragraph boundaries shall be used instead of section
120987 boundaries.

120988 **Move Forward to Next Paragraph**

120989 *Synopsis*: **[***count***]** }

120990 Move forward to the beginning of the next paragraph. This command shall be equivalent to the
120991 **]]** command, with the exception that paragraph boundaries shall be used instead of section
120992 boundaries.

120993 **Move to Specific Column Position**

120994 *Synopsis*: **[***count***]** |

120995 For the purposes of this command, lines that are too long for the current display and that have
120996 been folded shall be treated as having a single, 1–based, number of columns.

120997 If there are less than *count* columns in which characters from the current line are displayed on
120998 the screen, *count* shall be adjusted to be the last column in which any portion of the line is
120999 displayed on the screen.

121000 If used as a motion command:

121001 1. If the line is empty, or the cursor character is the same as the character on the *count*th
121002 column of the line, it shall be an error.

121003 2. If the cursor is before the *count*th column of the line, the text region shall be comprised of
121004 the current character, up to but not including the character on the *count*th column of the
121005 line.

121006 3. If the cursor is after the *count*th column of the line, the text region shall be from the
121007 character before the starting cursor up to and including the character on the *count*th
121008 column of the line.

121009 4. Any text copied to a buffer shall be in character mode.

121010 If not used as a motion command:

121011 *Current line*: Unchanged.

121012 *Current column*: Set to the last column in which any portion of the character that is displayed in
121013 the *count* column of the line is displayed.

121014 **Reverse Find Character**

121015 *Synopsis*: **[***count***]** **,**

121016 If the last **F**, **f**, **T**, or **t** command was **F**, **f**, **T**, or **t**, this command shall be equivalent to an **f**, **F**, **t**, or
121017 **T** command, respectively, with the specified *count* and the same search character.

121018 If there was no previous **F**, **f**, **T**, or **t** command, it shall be an error.

Repeat

Synopsis: **[***count***] .**

Repeat the last **!**, **<**, **>**, **A**, **C**, **D**, **I**, **J**, **O**, **P**, **R**, **S**, **X**, **Y**, **a**, **c**, **d**, **i**, **o**, **p**, **r**, **s**, **x**, **y**, or ~ command. It shall be an error if none of these commands have been executed. Commands (other than commands that enter text input mode) executed as a result of map expansions, shall not change the value of the last repeatable command.

Repeated commands with associated motion commands shall repeat the motion command as well; however, any specified *count* shall replace the *count*(s) that were originally specified to the repeated command or its associated motion command.

If the motion component of the repeated command is **f**, **F**, **t**, or **T**, the repeated command shall not set the remembered search character for the **;** and **,** commands.

If the repeated command is **p** or **P**, and the buffer associated with that command was a numeric buffer named with a number less than 9, the buffer associated with the repeated command shall be set to be the buffer named by the name of the previous buffer logically incremented by 1.

If the repeated character is a text input command, the input text associated with that command is repeated literally:

- Input characters are neither macro or abbreviation-expanded.
- Input characters are not interpreted in any special way with the exception that <newline>, <carriage-return>, and <control>-T behave as described in Input Mode Commands in vi (on page 3561).

Current line: Set as described for the repeated command.

Current column: Set as described for the repeated command.

Find Regular Expression

Synopsis: /

If the input line contains no non-<newline> characters, it shall be equivalent to a line containing only the last regular expression encountered. The enhanced regular expressions supported by *vi* are described in Regular Expressions in ex (on page 2875).

Otherwise, the line shall be interpreted as one or more regular expressions, optionally followed by an address offset or a *vi* **z** command.

If the regular expression is not the last regular expression on the line, or if a line offset or **z** command is specified, the regular expression shall be terminated by an unescaped '/' character, which shall not be used as part of the regular expression. If the regular expression is not the first regular expression on the line, it shall be preceded by zero or more <blank> characters, a <semicolon>, zero or more <blank> characters, and a leading '/' character, which shall not be interpreted as part of the regular expression. It shall be an error to precede any regular expression with any characters other than these.

Each search shall begin from the character after the first character of the last match (or, if it is the first search, after the cursor). If the **wrapscan** edit option is set, the search shall continue to the character before the starting cursor character; otherwise, to the end of the edit buffer. It shall be an error if any search fails to find a match, and an informational message to this effect shall be displayed.

An optional address offset (see Addressing in ex, on page 2845) can be specified after the last regular expression by including a trailing '/' character after the regular expression and

specifying the address offset. This offset shall be from the line containing the match for the last regular expression specified. It shall be an error if the line offset would indicate a line address less than 1 or greater than the last line in the edit buffer. An address offset of zero shall be supported. It shall be an error to follow the address offset with any other characters than <blank> characters.

If not used as a motion command, an optional **z** command (see Redraw Window, on page 3560) can be specified after the last regular expression by including a trailing `'/'` character after the regular expression, zero or more <blank> characters, a `'z'`, zero or more <blank> characters, an optional new **window** edit option value, zero or more <blank> characters, and a location character. The effect shall be as if the **z** command was executed after the **/** command. It shall be an error to follow the **z** command with any other characters than <blank> characters.

The remembered search direction shall be set to forward.

If used as a motion command:

1. It shall be an error if the last match references the same character in the edit buffer as the starting cursor.
2. If any address offset is specified, the last match shall be adjusted by the specified offset as described previously.
3. If the starting cursor is after the last match, then the locations of the starting cursor and the last match in the edit buffer shall be logically swapped.
4. If any address offset is specified, the text region shall consist of all lines containing characters from the starting cursor to the last match line, inclusive, and any text copied to a buffer shall be in line mode.
5. Otherwise, if the starting line is empty or the starting cursor is at or before the first non-<blank> non-<newline> of the starting line, and the last match line is empty or the last match starts at the first character of the last match line, the text region shall consist of all lines containing characters from the starting cursor to the line before the last match line, inclusive, and any text copied to a buffer shall be in line mode.
6. Otherwise, if the last match line is empty or the last match begins at a character at or before the first non-<blank> non-<newline> of the last match line, the region of text shall be from the current cursor to the last non-<newline> of the line before the last match line, inclusive, and any text copied to a buffer shall be in character mode.
7. Otherwise, the region of text shall be from the current cursor (inclusive), to the first character of the last match (exclusive), and any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: If a match is found, set to the last matched line plus the address offset, if any; otherwise, unchanged.

Current column: Set to the last column on which any portion of the first character in the last matched string is displayed, if a match is found; otherwise, unchanged.

Move to First Character in Line

Synopsis: `0` (zero)

Move to the first character on the current line. The character `'0'` shall not be interpreted as a command if it is immediately preceded by a digit.

If used as a motion command:

1. If the cursor character is the first character in the line, it shall be an error.
2. The text region shall be from the character before the cursor character up to and including the first character in the line.
3. Any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Unchanged.

Current column: The last column in which any portion of the first character in the line is displayed, or if the line is empty, unchanged.

Execute an ex Command

Synopsis: **:**

Execute one or more *ex* commands.

If any portion of the screen other than the last line of the screen was overwritten by any *ex* command (except **shell**), *vi* shall display a message indicating that it is waiting for an input from the user, and shall then read a character. This action may also be taken for other, unspecified reasons.

If the next character entered is a `':'`, another *ex* command shall be accepted and executed. Any other character shall cause the screen to be refreshed and *vi* shall return to command mode.

Current line: As specified for the *ex* command.

Current column: As specified for the *ex* command.

Repeat Find

Synopsis: **[***count***] ;**

This command shall be equivalent to the last **F**, **f**, **T**, or **t** command, with the specified *count*, and with the same search character used for the last **F**, **f**, **T**, or **t** command. If there was no previous **F**, **f**, **T**, or **t** command, it shall be an error.

Shift Left

Synopsis: **[***count***]** `<` *motion*

If the motion command is the **<** command repeated:

1. If there are less than *count* –1 lines after the current line in the edit buffer, it shall be an error.
2. The text region shall be from the current line, up to and including the next *count* –1 lines.

Shift any line in the text region specified by the *count* and motion command one shiftwidth (see the *ex* **shiftwidth** option) toward the start of the line, as described by the *ex* **<** command. The unshifted lines shall be copied to the unnamed buffer in line mode.

Current line: If the motion was from the current cursor position toward the end of the edit buffer, unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region specified by the motion command.

Current column: Set to non-<blank>.

Shift Right

Synopsis: `[count] > motion`

If the motion command is the **>** command repeated:

1. If there are less than *count* –1 lines after the current line in the edit buffer, it shall be an error.
2. The text region shall be from the current line, up to and including the next *count* –1 lines.

Shift any line with characters in the text region specified by the *count* and motion command one shiftwidth (see the *ex* **shiftwidth** option) away from the start of the line, as described by the *ex* **>** command. The unshifted lines shall be copied into the unnamed buffer in line mode.

Current line: If the motion was from the current cursor position toward the end of the edit buffer, unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region specified by the motion command.

Current column: Set to non-<blank>.

Scan Backwards for Regular Expression

Synopsis: `?`

Scan backwards; the **?** command shall be equivalent to the **/** command (see Find Regular Expression, on page 3543) with the following exceptions:

1. The input prompt shall be a `'?'`.
2. Each search shall begin from the character before the first character of the last match (or, if it is the first search, the character before the cursor character).
3. The search direction shall be from the cursor toward the beginning of the edit buffer, and the **wrapscan** edit option shall affect whether the search wraps to the end of the edit buffer and continues.
4. The remembered search direction shall be set to backward.

Execute

Synopsis: `@buffer`

If the *buffer* is specified as **@**, the last buffer executed shall be used. If no previous buffer has been executed, it shall be an error.

Behave as if the contents of the named buffer were entered as standard input. After each line of a line-mode buffer, and all but the last line of a character mode buffer, behave as if a <newline> were entered as standard input.

If an error occurs during this process, an error message shall be written, and no more characters resulting from the execution of this command shall be processed.

If a *count* is specified, behave as if that count were entered as user input before the characters from the **@** buffer were entered.

Current line: As specified for the individual commands.

Current column: As specified for the individual commands.

Reverse Case

Synopsis: **[***count***]** ~

Reverse the case of the current character and the next *count* –1 characters, such that lowercase characters that have uppercase counterparts shall be changed to uppercase characters, and uppercase characters that have lowercase counterparts shall be changed to lowercase characters, as prescribed by the current locale. No other characters shall be affected by this command.

If there are less than *count* –1 characters after the cursor in the edit buffer, *count* shall be adjusted to the number of characters after the cursor in the edit buffer minus 1.

For the purposes of this command, the next character after the last non-<newline> on the line shall be the next character in the edit buffer.

Current line: Set to the line including the (*count*–1)th character after the cursor.

Current column: Set to the last column in which any portion of the (*count*–1)th character after the cursor is displayed.

Append

Synopsis: **[***count***]** a

Enter text input mode after the current cursor position. No characters already in the edit buffer shall be affected by this command. A *count* shall cause the input text to be appended *count* –1 more times to the end of the input.

Current line/column: As specified for the text input commands (see Input Mode Commands in vi, on page 3561).

Append at End-of-Line

Synopsis: **[***count***]** A

This command shall be equivalent to the *vi* command:

$ **[** *count* **]** a

(see Append).

Move Backward to Preceding Word

Synopsis: **[***count***]** b

With the exception that words are used as the delimiter instead of bigwords, this command shall be equivalent to the **B** command.

121209 **Move Backward to Preceding Bigword**

121210 *Synopsis*: **[***count***]** B

121211 If the edit buffer is empty or the cursor is on the first character of the edit buffer, it shall be an
121212 error. If less than *count* bigwords begin between the cursor and the start of the edit buffer, *count*
121213 shall be adjusted to the number of bigword beginnings between the cursor and the start of the
121214 edit buffer.

121215 If used as a motion command:

121216 1. The text region shall be from the first character of the *count*th previous bigword beginning
121217 up to but not including the cursor character.

121218 2. Any text copied to a buffer shall be in character mode.

121219 If not used as a motion command:

121220 *Current line*: Set to the line containing the *current column*.

121221 *Current column*: Set to the last column upon which any part of the first character of the *count*th
121222 previous bigword is displayed.

121223 **Change**

121224 *Synopsis*: **[***buffer***][***count***]** c *motion*

121225 If the motion command is the **c** command repeated:

121226 1. The buffer text shall be in line mode.

121227 2. If there are less than *count* –1 lines after the current line in the edit buffer, it shall be an
121228 error.

121229 3. The text region shall be from the current line up to and including the next *count* –1 lines.

121230 Otherwise, the buffer text mode and text region shall be as specified by the motion command.

121231 The replaced text shall be copied into *buffer*, if specified, and into the unnamed buffer. If the text
121232 to be replaced contains characters from more than a single line, or the buffer text is in line mode,
121233 the replaced text shall be copied into the numeric buffers as well.

121234 If the buffer text is in line mode:

121235 1. Any lines that contain characters in the region shall be deleted, and the editor shall enter
121236 text input mode at the beginning of a new line which shall replace the first line deleted.

121237 2. If the **autoindent** edit option is set, **autoindent** characters equal to the **autoindent**
121238 characters on the first line deleted shall be inserted as if entered by the user.

121239 Otherwise, if characters from more than one line are in the region of text:

121240 1. The text shall be deleted.

121241 2. Any text remaining in the last line in the text region shall be appended to the first line in
121242 the region, and the last line in the region shall be deleted.

121243 3. The editor shall enter text input mode after the last character not deleted from the first
121244 line in the text region, if any; otherwise, on the first column of the first line in the region.

Otherwise:

1. If the glyph for `'$'` is smaller than the region, the end of the region shall be marked with a `'$'`.
2. The editor shall enter text input mode, overwriting the region of text.

Current line/column: As specified for the text input commands (see Input Mode Commands in vi, on page 3561).

Change to End-of-Line

Synopsis: **[***buffer***][***count***]** `C`

This command shall be equivalent to the *vi* command:

[*buffer***][***count***]** `c$`

See the **c** command.

Delete

Synopsis: **[***buffer***][***count***]** `d` *motion*

If the motion command is the **d** command repeated:

1. The buffer text shall be in line mode.
2. If there are less than *count* –1 lines after the current line in the edit buffer, it shall be an error.
3. The text region shall be from the current line up to and including the next *count* –1 lines.

Otherwise, the buffer text mode and text region shall be as specified by the motion command.

If in open mode, and the current line is deleted, and the line remains on the display, an `'@'` character shall be displayed as the first glyph of that line.

Delete the region of text into *buffer*, if specified, and into the unnamed buffer. If the text to be deleted contains characters from more than a single line, or the buffer text is in line mode, the deleted text shall be copied into the numeric buffers, as well.

Current line: Set to the first text region line that appears in the edit buffer, unless that line has been deleted, in which case it shall be set to the last line in the edit buffer, or line 1 if the edit buffer is empty.

Current column:

1. If the line is empty, set to column position 1.
2. Otherwise, if the buffer text is in line mode or the motion was from the cursor toward the end of the edit buffer:
   a. If a character from the current line is displayed in the current column, set to the last column that displays any portion of that character.
   b. Otherwise, set to the last column in which any portion of any character in the line is displayed.
3. Otherwise, if a character is displayed in the column that began the text region, set to the last column that displays any portion of that character.

4. Otherwise, set to the last column in which any portion of any character in the line is displayed.

Delete to End-of-Line

Synopsis: **[***buffer***]** D

Delete the text from the current position to the end of the current line; equivalent to the *vi* command:

[*buffer***]** d$

Move to End-of-Word

Synopsis: **[***count***]** e

With the exception that words are used instead of bigwords as the delimiter, this command shall be equivalent to the **E** command.

Move to End-of-Bigword

Synopsis: **[***count***]** E

If the edit buffer is empty it shall be an error. If less than *count* bigwords end between the cursor and the end of the edit buffer, *count* shall be adjusted to the number of bigword endings between the cursor and the end of the edit buffer.

If used as a motion command:

1. The text region shall be from the last character of the *count*th next bigword up to and including the cursor character.
2. Any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Set to the line containing the current column.

Current column: Set to the last column upon which any part of the last character of the *count*th next bigword is displayed.

Find Character in Current Line (Forward)

Synopsis: **[***count***]** f *character*

It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

If used as a motion command:

1. The text range shall be from the cursor character up to and including the *count*th occurrence of the specified character after the cursor.
2. Any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Unchanged.

Current column: Set to the last column in which any portion of the *count*th occurrence of the specified character after the cursor appears in the line.

121317 **Find Character in Current Line (Reverse)**

121318 *Synopsis*: **[***count***]** `F` *character*

121319 It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

121320 If used as a motion command:

121321 1. The text region shall be from the *count*th occurrence of the specified character before the
121322 cursor, up to, but not including the cursor character.

121323 2. Any text copied to a buffer shall be in character mode.

121324 If not used as a motion command:

121325 *Current line*: Unchanged.

121326 *Current column*: Set to the last column in which any portion of the *count*th occurrence of the
121327 specified character before the cursor appears in the line.

121328 **Move to Line**

121329 *Synopsis*: **[***count***]** `G`

121330 If *count* is not specified, it shall default to the last line of the edit buffer. If *count* is greater than
121331 the last line of the edit buffer, it shall be an error.

121332 If used as a motion command:

121333 1. The text region shall be from the cursor line up to and including the specified line.

121334 2. Any text copied to a buffer shall be in line mode.

121335 If not used as a motion command:

121336 *Current line*: Set to *count* if *count* is specified; otherwise, the last line.

121337 *Current column*: Set to non-<blank>.

121338 **Move to Top of Screen**

121339 *Synopsis*: **[***count***]** `H`

121340 If the beginning of the line *count* greater than the first line of which any portion appears on the
121341 display does not exist, it shall be an error.

121342 If used as a motion command:

121343 1. If in open mode, the text region shall be the current line.

121344 2. Otherwise, the text region shall be from the starting line up to and including (the first line
121345 of the display + *count* –1).

121346 3. Any text copied to a buffer shall be in line mode.

121347 If not used as a motion command:

121348 If in open mode, this command shall set the current column to non-<blank> and do nothing else.

121349 Otherwise, it shall set the current line and current column as follows.

121350 *Current line*: Set to (the first line of the display + *count* –1).

121351 *Current column*: Set to non-<blank>.

121352 **Insert Before Cursor**

121353 *Synopsis*: **[***count***]** i

121354 Enter text input mode before the current cursor position. No characters already in the edit buffer
121355 shall be affected by this command. A *count* shall cause the input text to be appended *count* –1
121356 more times to the end of the input.

121357 *Current line/column*: As specified for the text input commands (see Input Mode Commands in vi, on page 3561).

121359 **Insert at Beginning of Line**

121360 *Synopsis*: **[***count***]** I

121361 This command shall be equivalent to the *vi* command ^[*count*]**i**.

121362 **Join**

121363 *Synopsis*: **[***count***]** J

121364 If the current line is the last line in the edit buffer, it shall be an error.

121365 This command shall be equivalent to the *ex* **join** command with no addresses, and an *ex*
121366 command *count* value of 1 if *count* was not specified or if a *count* of 1 was specified, and an *ex*
121367 command *count* value of *count* –1 for any other value of *count*, except that the current line and
121368 column shall be set as follows.

121369 *Current line*: Unchanged.

121370 *Current column*: The last column in which any portion of the character following the last
121371 character in the initial line is displayed, or the last non-<newline> in the line if no characters
121372 were appended.

121373 **Move to Bottom of Screen**

121374 *Synopsis*: **[***count***]** L

121375 If the beginning of the line *count* less than the last line of which any portion appears on the
121376 display does not exist, it shall be an error.

121377 If used as a motion command:

121378 1. If in open mode, the text region shall be the current line.

121379 2. Otherwise, the text region shall include all lines from the starting cursor line to (the last
121380 line of the display –(*count* –1)).

121381 3. Any text copied to a buffer shall be in line mode.

121382 If not used as a motion command:

121383 1. If in open mode, this command shall set the current column to non-<blank> and do
121384 nothing else.

121385 2. Otherwise, it shall set the current line and current column as follows.

121386 *Current line*: Set to (the last line of the display –(*count* –1)).

121387 *Current column*: Set to non-<blank>.

121388 **Mark Position**

121389 *Synopsis*: `m` *`letter`*

121390 This command shall be equivalent to the *ex* **mark** command with the specified character as an
121391 argument.

121392 **Move to Middle of Screen**

121393 *Synopsis*: `M`

121394 The middle line of the display shall be calculated as follows:

121395 `(the top line of the display) + (((number of lines displayed) +1) /2) -1`

121396 If used as a motion command:

121397 1. If in open mode, the text region shall be the current line.

121398 2. Otherwise, the text region shall include all lines from the starting cursor line up to and
121399 including the middle line of the display.

121400 3. Any text copied to a buffer shall be in line mode.

121401 If not used as a motion command:

121402 If in open mode, this command shall set the current column to non-<blank> and do nothing else.

121403 Otherwise, it shall set the current line and current column as follows.

121404 *Current line*: Set to the middle line of the display.

121405 *Current column*: Set to non-<blank>.

121406 **Repeat Regular Expression Find (Forward)**

121407 *Synopsis*: `n`

121408 If the remembered search direction was forward, the **n** command shall be equivalent to the *vi* **/**
121409 command with no characters entered by the user. Otherwise, it shall be equivalent to the *vi* **?**
121410 command with no characters entered by the user.

121411 If the **n** command is used as a motion command for the **!** command, the editor shall not enter
121412 text input mode on the last line on the screen, and shall behave as if the user entered a single
121413 `'!'` character as the text input.

121414 **Repeat Regular Expression Find (Reverse)**

121415 *Synopsis*: `N`

121416 Scan for the next match of the last pattern given to **/** or **?**, but in the reverse direction; this is the
121417 reverse of **n**.

121418 If the remembered search direction was forward, the **N** command shall be equivalent to the *vi* **?**
121419 command with no characters entered by the user. Otherwise, it shall be equivalent to the *vi* **/**
121420 command with no characters entered by the user. If the **N** command is used as a motion
121421 command for the **!** command, the editor shall not enter text input mode on the last line on the
121422 screen, and shall behave as if the user entered a single **!** character as the text input.

121423 **Insert Empty Line Below**

121424 *Synopsis*: o

121425 Enter text input mode in a new line appended after the current line. A *count* shall cause the input
121426 text to be appended *count* –1 more times to the end of the already added text, each time starting
121427 on a new, appended line.

121428 *Current line/column*: As specified for the text input commands (see Input Mode Commands in vi,
121429 on page 3561).

121430 **Insert Empty Line Above**

121431 *Synopsis*: O

121432 Enter text input mode in a new line inserted before the current line. A *count* shall cause the input
121433 text to be appended *count* –1 more times to the end of the already added text, each time starting
121434 on a new, appended line.

121435 *Current line/column*: As specified for the text input commands (see Input Mode Commands in vi,
121436 on page 3561).

121437 **Put from Buffer Following**

121438 *Synopsis*: **[***buffer***]** p

121439 If no *buffer* is specified, the unnamed buffer shall be used.

121440 If the buffer text is in line mode, the text shall be appended below the current line, and each line
121441 of the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be
121442 appended *count* –1 more times to the end of the already added text, each time starting on a new,
121443 appended line.

121444 If the buffer text is in character mode, the text shall be appended into the current line after the
121445 cursor, and each line of the buffer other than the first and last shall become a new line in the edit
121446 buffer. A *count* shall cause the buffer text to be appended *count* –1 more times to the end of the
121447 already added text, each time starting after the last added character.

121448 *Current line*: If the buffer text is in line mode, set the line to line +1; otherwise, unchanged.

121449 *Current column*: If the buffer text is in line mode:

121450 1. If there is a non-<blank> in the first line of the buffer, set to the last column on which any
121451 portion of the first non-<blank> in the line is displayed.

121452 2. If there is no non-<blank> in the first line of the buffer, set to the last column on which
121453 any portion of the last non-<newline> in the first line of the buffer is displayed.

121454 If the buffer text is in character mode:

121455 1. If the text in the buffer is from more than a single line, then set to the last column on
121456 which any portion of the first character from the buffer is displayed.

121457 2. Otherwise, if the buffer is the unnamed buffer, set to the last column on which any
121458 portion of the last character from the buffer is displayed.

121459 3. Otherwise, set to the first column on which any portion of the first character from the
121460 buffer is displayed.

121461 **Put from Buffer Before**

121462 *Synopsis*: **[***buffer***]** P

121463 If no *buffer* is specified, the unnamed buffer shall be used.

121464 If the buffer text is in line mode, the text shall be inserted above the current line, and each line of
121465 the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be
121466 appended *count* –1 more times to the end of the already added text, each time starting on a new,
121467 appended line.

121468 If the buffer text is in character mode, the text shall be inserted into the current line before the
121469 cursor, and each line of the buffer other than the first and last shall become a new line in the edit
121470 buffer. A *count* shall cause the buffer text to be appended *count* –1 more times to the end of the
121471 already added text, each time starting after the last added character.

121472 *Current line*: Unchanged.

121473 *Current column*: If the buffer text is in line mode:

121474 1. If there is a non-<blank> in the first line of the buffer, set to the last column on which any
121475 portion of that character is displayed.

121476 2. If there is no non-<blank> in the first line of the buffer, set to the last column on which
121477 any portion of the last non-<newline> in the first line of the buffer is displayed.

121478 If the buffer text is in character mode:

121479 1. If the text in the buffer is from more than a single line, then set to the last column on
121480 which any portion of the first character from the buffer is displayed.

121481 2. Otherwise, if the buffer is the unnamed buffer, set to the last column on which any
121482 portion of the last character from the buffer is displayed.

121483 3. Otherwise, set to the first column on which any portion of the first character from the
121484 buffer is displayed.

121485 **Enter ex Mode**

121486 *Synopsis*: Q

121487 Leave visual or open mode and enter *ex* command mode.

121488 *Current line*: Unchanged.

121489 *Current column*: Unchanged.

121490 **Replace Character**

121491 *Synopsis*: **[***count***]** r *character*

121492 Replace the *count* characters at and after the cursor with the specified character. If there are less
121493 than *count* non-<newline> characters at and after the cursor on the line, it shall be an error.

121494 If character is <control>-V, any next character other than the <newline> shall be stripped of any
121495 special meaning and used as a literal character.

121496 If character is <ESC>, no replacement shall be made and the current line and current column
121497 shall be unchanged.

121498 If character is <carriage-return> or <newline>, *count* new lines shall be appended to the current
121499 line. All but the last of these lines shall be empty. *count* characters at and after the cursor shall be

discarded, and any remaining characters after the cursor in the current line shall be moved to the last of the new lines. If the **autoindent** edit option is set, they shall be preceded by the same number of **autoindent** characters found on the line from which the command was executed.

Current line: Unchanged unless the replacement character is a <carriage-return> or <newline>, in which case it shall be set to line + *count*.

Current column: Set to the last column position on which a portion of the last replaced character is displayed, or if the replacement character caused new lines to be created, set to non-<blank>.

Replace Characters

Synopsis: `R`

Enter text input mode at the current cursor position possibly replacing text on the current line. A *count* shall cause the input text to be appended *count* –1 more times to the end of the input.

Current line/column: As specified for the text input commands (see Input Mode Commands in vi, on page 3561).

Substitute Character

Synopsis: **[***buffer***][***count***]** s

This command shall be equivalent to the *vi* command:

[*buffer***][***count***]** c<space>

Substitute Lines

Synopsis: **[***buffer***][***count***]** S

This command shall be equivalent to the *vi* command:

[*buffer***][***count***]** c_

Move Cursor to Before Character (Forward)

Synopsis: **[***count***]** t *character*

It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

If used as a motion command:

1. The text region shall be from the cursor up to but not including the *count*th occurrence of the specified character after the cursor.
2. Any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Unchanged.

Current column: Set to the last column in which any portion of the character before the *count*th occurrence of the specified character after the cursor appears in the line.

Move Cursor to After Character (Reverse)

Synopsis: `[`*`count`*`] T `*`character`*

It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

If used as a motion command:

1. If the character before the cursor is the specified character, it shall be an error.
2. The text region shall be from the character before the cursor up to but not including the *count*th occurrence of the specified character before the cursor.
3. Any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Unchanged.

Current column: Set to the last column in which any portion of the character after the *count*th occurrence of the specified character before the cursor appears in the line.

Undo

Synopsis: `u`

This command shall be equivalent to the *ex* **undo** command except that the current line and current column shall be set as follows:

Current line: Set to the first line added or changed if any; otherwise, move to the line preceding any deleted text if one exists; otherwise, move to line 1.

Current column: If undoing an *ex* command, set to the first non-<blank>.

Otherwise, if undoing a text input command:

1. If the command was a **C**, **c**, **O**, **o**, **R**, **S**, or **s** command, the current column shall be set to the value it held when the text input command was entered.
2. Otherwise, set to the last column in which any portion of the first character after the deleted text is displayed, or, if no non-<newline> characters follow the text deleted from this line, set to the last column in which any portion of the last non-<newline> in the line is displayed, or 1 if the line is empty.

Otherwise, if a single line was modified (that is, not added or deleted) by the **u** command:

1. If text was added or changed, set to the last column in which any portion of the first character added or changed is displayed.
2. If text was deleted, set to the last column in which any portion of the first character after the deleted text is displayed, or, if no non-<newline> characters follow the deleted text, set to the last column in which any portion of the last non-<newline> in the line is displayed, or 1 if the line is empty.

Otherwise, set to non-<blank>.

121566 **Undo Current Line**

121567 *Synopsis*: `U`

121568 Restore the current line to its state immediately before the most recent time that it became the
121569 current line.

121570 *Current line*: Unchanged.

121571 *Current column*: Set to the first column in the line in which any portion of the first character in
121572 the line is displayed.

121573 **Move to Beginning of Word**

121574 *Synopsis*: **[***count***]** `w`

121575 With the exception that words are used as the delimiter instead of bigwords, this command shall
121576 be equivalent to the **W** command.

121577 **Move to Beginning of Bigword**

121578 *Synopsis*: **[***count***]** `W`

121579 If the edit buffer is empty, it shall be an error. If there are less than *count* bigwords between the
121580 cursor and the end of the edit buffer, *count* shall be adjusted to move the cursor to the last
121581 bigword in the edit buffer.

121582 If used as a motion command:

121583 1. If the associated command is **c**, *count* is 1, and the cursor is on a <blank>, the region of
121584 text shall be the current character and no further action shall be taken.

121585 2. If there are less than *count* bigwords between the cursor and the end of the edit buffer,
121586 then the command shall succeed, and the region of text shall include the last character of
121587 the edit buffer.

121588 3. If there are <blank> characters or an end-of-line that precede the *count*th bigword, and the
121589 associated command is **c**, the region of text shall be up to and including the last character
121590 before the preceding <blank> characters or end-of-line.

121591 4. If there are <blank> characters or an end-of-line that precede the bigword, and the
121592 associated command is **d** or **y**, the region of text shall be up to and including the last
121593 <blank> before the start of the bigword or end-of-line.

121594 5. Any text copied to a buffer shall be in character mode.

121595 If not used as a motion command:

121596 1. If the cursor is on the last character of the edit buffer, it shall be an error.

121597 *Current line*: Set to the line containing the current column.

121598 *Current column*: Set to the last column in which any part of the first character of the *count*th next
121599 bigword is displayed.

121600 **Delete Character at Cursor**

121601 *Synopsis*: **[***buffer***][***count***]** x

121602 Delete the *count* characters at and after the current character into *buffer*, if specified, and into the
121603 unnamed buffer.

121604 If the line is empty, it shall be an error. If there are less than *count* non-<newline> characters at
121605 and after the cursor on the current line, *count* shall be adjusted to the number of non-<newline>
121606 characters at and after the cursor.

121607 *Current line*: Unchanged.

121608 *Current column*: If the line is empty, set to column position 1. Otherwise, if there were *count* or
121609 less non-<newline> characters at and after the cursor on the current line, set to the last column
121610 that displays any part of the last non-<newline> of the line. Otherwise, unchanged.

121611 **Delete Character Before Cursor**

121612 *Synopsis*: **[***buffer***][***count***]** X

121613 Delete the *count* characters before the current character into *buffer*, if specified, and into the
121614 unnamed buffer.

121615 If there are no characters before the current character on the current line, it shall be an error. If
121616 there are less than *count* previous characters on the current line, *count* shall be adjusted to the
121617 number of previous characters on the line.

121618 *Current line*: Unchanged.

121619 *Current column*: Set to (current column – the width of the deleted characters).

121620 **Yank**

121621 *Synopsis*: **[***buffer***][***count***]** y *motion*

121622 Copy (yank) the region of text into *buffer*, if specified, and into the unnamed buffer.

121623 If the motion command is the **y** command repeated:

121624 1. The buffer shall be in line mode.

121625 2. If there are less than *count* –1 lines after the current line in the edit buffer, it shall be an
121626 error.

121627 3. The text region shall be from the current line up to and including the next *count* –1 lines.

121628 Otherwise, the buffer text mode and text region shall be as specified by the motion command.

121629 *Current line*: If the motion was from the current cursor position toward the end of the edit buffer,
121630 unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region
121631 specified by the motion command.

121632 *Current column*:

121633 1. If the motion was from the current cursor position toward the end of the edit buffer,
121634 unchanged.

121635 2. Otherwise, if the current line is empty, set to column position 1.

121636 3. Otherwise, set to the last column that displays any part of the first character in the file
121637 that is part of the text region specified by the motion command.

121638 **Yank Current Line**

121639 *Synopsis*: **[***buffer***][***count***]** Y

121640 This command shall be equivalent to the *vi* command:

121641 **[***buffer***][***count***]** y_

121642 **Redraw Window**

121643 If in open mode, the **z** command shall have the Synopsis:

121644 *Synopsis*: **[***count***]** z

121645 If *count* is not specified, it shall default to the **window** edit option –1. The **z** command shall be
121646 equivalent to the *ex* **z** command, with a type character of **=** and a *count* of *count* –2, except that
121647 the current line and current column shall be set as follows, and the **window** edit option shall not
121648 be affected. If the calculation for the *count* argument would result in a negative number, the
121649 *count* argument to the *ex* **z** command shall be zero. A blank line shall be written after the last line
121650 is written.

121651 *Current line*: Unchanged.

121652 *Current column*: Unchanged.

121653 If not in open mode, the **z** command shall have the following Synopsis:

121654 *Synopsis*: **[***line***]** z **[***count***]** *character*

121655 If *line* is not specified, it shall default to the current line. If *line* is specified, but is greater than the
121656 number of lines in the edit buffer, it shall default to the number of lines in the edit buffer.

121657 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in the
121658 *ex* **window** command), and the screen shall be redrawn.

121659 *line* shall be placed as specified by the following characters:

121660 <newline>, <carriage-return>
121661 Place the beginning of the line on the first line of the display.

121662 **.** Place the beginning of the line in the center of the display. The middle line of the display
121663 shall be calculated as described for the **M** command.

121664 – Place an unspecified portion of the line on the last line of the display.

121665 + If *line* was specified, equivalent to the <newline> case. If *line* was not specified, display a
121666 screen where the first line of the display shall be (current last line) +1. If there are no lines
121667 after the last line in the display, it shall be an error.

121668 ^ If *line* was specified, display a screen where the last line of the display shall contain an
121669 unspecified portion of the first line of a display that had an unspecified portion of the
121670 specified line on the last line of the display. If this calculation results in a line before the
121671 beginning of the edit buffer, display the first screen of the edit buffer.

121672 Otherwise, display a screen where the last line of the display shall contain an unspecified
121673 portion of (current first line –1). If this calculation results in a line before the beginning of
121674 the edit buffer, it shall be an error.

Current line: If *line* and the `'^'` character were specified:

1. If the first screen was displayed as a result of the command attempting to display lines before the beginning of the edit buffer: if the first screen was already displayed, unchanged; otherwise, set to (current first line –1).
2. Otherwise, set to the last line of the display.

If *line* and the `'+'` character were specified, set to the first line of the display.

Otherwise, if *line* was specified, set to *line*.

Otherwise, unchanged.

Current column: Set to non-<blank>.

Exit

Synopsis: `ZZ`

This command shall be equivalent to the *ex* **xit** command with no addresses, trailing **!**, or filename (see the *ex* **xit** command).

Input Mode Commands in vi

In text input mode, the current line shall consist of zero or more of the following categories, plus the terminating <newline>:

1. Characters preceding the text input entry point

   Characters in this category shall not be modified during text input mode.

2. **autoindent** characters

   autoindent characters shall be automatically inserted into each line that is created in text input mode, either as a result of entering a <newline> or <carriage-return> while in text input mode, or as an effect of the command itself; for example, **O** or **o** (see the *ex* **autoindent** command), as if entered by the user.

   It shall be possible to erase **autoindent** characters with the <control>-D command; it is unspecified whether they can be erased by <control>-H, <control>-U, and <control>-W characters. Erasing any **autoindent** character turns the glyph into erase-columns and deletes the character from the edit buffer, but does not change its representation on the screen.

3. Text input characters

   Text input characters are the characters entered by the user. Erasing any text input character turns the glyph into erase-columns and deletes the character from the edit buffer, but does not change its representation on the screen.

   Each text input character entered by the user (that does not have a special meaning) shall be treated as follows:

   a. The text input character shall be appended to the last character in the edit buffer from the first, second, or third categories.

   b. If there are no erase-columns on the screen, the text input command was the **R** command, and characters in the fifth category from the original line follow the cursor, the next such character shall be deleted from the edit buffer. If the **slowopen** edit option is not set, the corresponding glyph on the screen shall

become erase-columns.

c. If there are erase-columns on the screen, as many columns as they occupy, or as are necessary, shall be overwritten to display the text input character. (If only part of a multi-column glyph is overwritten, the remainder shall be left on the screen, and continue to be treated as erase-columns; it is unspecified whether the remainder of the glyph is modified in any way.)

d. If additional display line columns are needed to display the text input character:

   i. If the **slowopen** edit option is set, the text input characters shall be displayed on subsequent display line columns, overwriting any characters displayed in those columns.

   ii. Otherwise, any characters currently displayed on or after the column on the display line where the text input character is to be displayed shall be pushed ahead the number of display line columns necessary to display the rest of the text input character.

4. Erase-columns

   Erase-columns are not logically part of the edit buffer, appearing only on the screen, and may be overwritten on the screen by subsequent text input characters. When text input mode ends, all erase-columns shall no longer appear on the screen.

   Erase-columns are initially the region of text specified by the **c** command (see Change, on page 3548); however, erasing **autoindent** or text input characters causes the glyphs of the erased characters to be treated as erase-columns.

5. Characters following the text region for the **c** command, or the text input entry point for all other commands

   Characters in this category shall not be modified during text input mode, except as specified in category 3.b. for the **R** text input command, or as <blank> characters deleted when a <newline> or <carriage-return> is entered.

It is unspecified whether it is an error to attempt to erase past the beginning of a line that was created by the entry of a <newline> or <carriage-return> during text input mode. If it is not an error, the editor shall behave as if the erasing character was entered immediately after the last text input character entered on the previous line, and all of the non-<newline> characters on the current line shall be treated as erase-columns.

When text input mode is entered, or after a text input mode character is entered (except as specified for the special characters below), the cursor shall be positioned as follows:

1. On the first column that displays any part of the first erase-column, if one exists
2. Otherwise, if the **slowopen** edit option is set, on the first display line column after the last character in the first, second, or third categories, if one exists
3. Otherwise, the first column that displays any part of the first character in the fifth category, if one exists
4. Otherwise, the display line column after the last character in the first, second, or third categories, if one exists
5. Otherwise, on column position 1

The characters that are updated on the screen during text input mode are unspecified, other than that the last text input character shall always be updated, and, if the **slowopen** edit option is not

121758 set, the current cursor character shall always be updated.

121759 The following specifications are for command characters entered during text input mode.

121760 **NUL**

121761 *Synopsis*: `NUL`

121762 If the first character of the text input is a NUL, the most recently input text shall be input as if
121763 entered by the user, and then text input mode shall be exited. The text shall be input literally;
121764 that is, characters are neither macro or abbreviation expanded, nor are any characters interpreted
121765 in any special manner. It is unspecified whether implementations shall support more than 256
121766 bytes of remembered input text.

121767 **<control>-D**

121768 *Synopsis*: `<control>-D`

121769 The <control>-D character shall have no special meaning when in text input mode for a line-
121770 oriented command (see Command Descriptions in vi, on page 3527).

121771 This command need not be supported on block-mode terminals.

121772 If the cursor does not follow an **autoindent** character, or an **autoindent** character and a `'0'` or
121773 `'^'` character:

121774 1. If the cursor is in column position 1, the <control>-D character shall be discarded and no
121775 further action taken.

121776 2. Otherwise, the <control>-D character shall have no special meaning.

121777 If the last input character was a `'0'`, the cursor shall be moved to column position 1.

121778 Otherwise, if the last input character was a `'^'`, the cursor shall be moved to column position 1.
121779 In addition, the **autoindent** level for the next input line shall be derived from the same line from
121780 which the **autoindent** level for the current input line was derived.

121781 Otherwise, the cursor shall be moved back to the column after the previous shiftwidth (see the
121782 *ex* **shiftwidth** command) boundary.

121783 All of the glyphs on columns between the starting cursor position and (inclusively) the ending
121784 cursor position shall become erase-columns as described in Input Mode Commands in vi (on
121785 page 3561).

121786 *Current line*: Unchanged.

121787 *Current column*: Set to 1 if the <control>-D was preceded by a `'^'` or `'0'`; otherwise, set to
121788 (column –1) –((column –2) % **shiftwidth**).

121789 **<control>-H**

121790 *Synopsis*: `<control>-H`

121791 If in text input mode for a line-oriented command, and there are no characters to erase, text
121792 input mode shall be terminated, no further action shall be done for this command, and the
121793 current line and column shall be unchanged.

121794 If there are characters other than **autoindent** characters that have been input on the current line
121795 before the cursor, the cursor shall move back one character.

121796 Otherwise, if there are **autoindent** characters on the current line before the cursor, it is

implementation-defined whether the <control>-H command is an error or if the cursor moves back one **autoindent** character.

Otherwise, if the cursor is in column position 1 and there are previous lines that have been input, it is implementation-defined whether the <control>-H command is an error or if it is equivalent to entering <control>-H after the last input character on the previous input line.

Otherwise, it shall be an error.

All of the glyphs on columns between the starting cursor position and (inclusively) the ending cursor position shall become erase-columns as described in Input Mode Commands in vi (on page 3561).

The current erase character (see *stty*) shall cause an equivalent action to the <control>-H command, unless the previously inserted character was a <backslash>, in which case it shall be as if the literal current erase character had been inserted instead of the <backslash>.

Current line: Unchanged, unless previously input lines are erased, in which case it shall be set to line –1.

Current column: Set to the first column that displays any portion of the character backed up over.

<newline>

Synopsis:

```
<newline>
<carriage-return>
<control>-J
<control>-M
```

If input was part of a line-oriented command, text input mode shall be terminated and the command shall continue execution with the input provided.

Otherwise, terminate the current line. If there are no characters other than **autoindent** characters on the line, all characters on the line shall be discarded. Otherwise, it is unspecified whether the **autoindent** characters in the line are modified by entering these characters.

Continue text input mode on a new line appended after the current line. If the **slowopen** edit option is set, the lines on the screen below the current line shall not be pushed down, but the first of them shall be cleared and shall appear to be overwritten. Otherwise, the lines of the screen below the current line shall be pushed down.

If the **autoindent** edit option is set, an appropriate number of **autoindent** characters shall be added as a prefix to the line as described by the *ex* **autoindent** edit option.

All columns after the cursor that are erase-columns (as described in Input Mode Commands in vi, on page 3561) shall be discarded.

If the **autoindent** edit option is set, all <blank> characters immediately following the cursor shall be discarded.

All remaining characters after the cursor shall be transferred to the new line, positioned after any **autoindent** characters.

Current line: Set to current line +1.

Current column: Set to the first column that displays any portion of the first character after the **autoindent** characters on the new line, if any, or the first column position after the last **autoindent** character, if any, or column position 1.

121838 **<control>-T**

121839 *Synopsis*: `<control>-T`

121840 The <control>-T character shall have no special meaning when in text input mode for a line-
121841 oriented command (see Command Descriptions in vi, on page 3527).

121842 This command need not be supported on block-mode terminals.

121843 Behave as if the user entered the minimum number of <blank> characters necessary to move the
121844 cursor forward to the column position after the next **shiftwidth** (see the *ex* **shiftwidth**
121845 command) boundary.

121846 *Current line*: Unchanged.

121847 *Current column*: Set to *column* + **shiftwidth** – ((column –1) % **shiftwidth**).

121848 **<control>-U**

121849 *Synopsis*: `<control>-U`

121850 If there are characters other than **autoindent** characters that have been input on the current line
121851 before the cursor, the cursor shall move to the first character input after the **autoindent**
121852 characters.

121853 Otherwise, if there are **autoindent** characters on the current line before the cursor, it is
121854 implementation-defined whether the <control>-U command is an error or if the cursor moves to
121855 the first column position on the line.

121856 Otherwise, if the cursor is in column position 1 and there are previous lines that have been
121857 input, it is implementation-defined whether the <control>-U command is an error or if it is
121858 equivalent to entering <control>-U after the last input character on the previous input line.

121859 Otherwise, it shall be an error.

121860 All of the glyphs on columns between the starting cursor position and (inclusively) the ending
121861 cursor position shall become erase-columns as described in Input Mode Commands in vi (on
121862 page 3561).

121863 The current *kill* character (see *stty*) shall cause an equivalent action to the <control>-U command,
121864 unless the previously inserted character was a <backslash>, in which case it shall be as if the
121865 literal current *kill* character had been inserted instead of the <backslash>.

121866 *Current line*: Unchanged, unless previously input lines are erased, in which case it shall be set to
121867 line –1.

121868 *Current column*: Set to the first column that displays any portion of the last character backed up
121869 over.

121870 **<control>-V**

121871 *Synopsis*: `<control>-V`
121872 `<control>-Q`

121873 Allow the entry of any subsequent character, other than <control>-J or the <newline>, as a literal
121874 character, removing any special meaning that it may have to the editor in text input mode. If a
121875 <control>-V or <control>-Q is entered before a <control>-J or <newline>, the <control>-V or
121876 <control>-Q character shall be discarded, and the <control>-J or <newline> shall behave as
121877 described in the <newline> command character during input mode.

121878 For purposes of the display only, the editor shall behave as if a `'^'` character was entered, and

the cursor shall be positioned as if overwriting the '^' character. When a subsequent character is entered, the editor shall behave as if that character was entered instead of the original <control>-V or <control>-Q character.

Current line: Unchanged.

Current column: Unchanged.

<control>-W

Synopsis: `<control>-W`

If there are characters other than **autoindent** characters that have been input on the current line before the cursor, the cursor shall move back over the last word preceding the cursor (including any <blank> characters between the end of the last word and the current cursor); the cursor shall not move to before the first character after the end of any **autoindent** characters.

Otherwise, if there are **autoindent** characters on the current line before the cursor, it is implementation-defined whether the <control>-W command is an error or if the cursor moves to the first column position on the line.

Otherwise, if the cursor is in column position 1 and there are previous lines that have been input, it is implementation-defined whether the <control>-W command is an error or if it is equivalent to entering <control>-W after the last input character on the previous input line.

Otherwise, it shall be an error.

All of the glyphs on columns between the starting cursor position and (inclusively) the ending cursor position shall become erase-columns as described in Input Mode Commands in vi (on page 3561).

Current line: Unchanged, unless previously input lines are erased, in which case it shall be set to line –1.

Current column: Set to the first column that displays any portion of the last character backed up over.

<ESC>

Synopsis: `<ESC>`

If input was part of a line-oriented command:

1. If *interrupt* was entered, text input mode shall be terminated and the editor shall return to command mode. The terminal shall be alerted.
2. If <ESC> was entered, text input mode shall be terminated and the command shall continue execution with the input provided.

Otherwise, terminate text input mode and return to command mode.

Any **autoindent** characters entered on newly created lines that have no other non-<newline> characters shall be deleted.

Any leading **autoindent** and <blank> characters on newly created lines shall be rewritten to be the minimum number of <blank> characters possible.

The screen shall be redisplayed as necessary to match the contents of the edit buffer.

Current line: Unchanged.

Current column:

1. If there are text input characters on the current line, the column shall be set to the last column where any portion of the last text input character is displayed.
2. Otherwise, if a character is displayed in the current column, unchanged.
3. Otherwise, set to column position 1.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

When an unrecoverable error is encountered it shall be equivalent to a SIGHUP asynchronous event.

Otherwise, when an error is encountered, the editor shall behave as specified in Command Descriptions in vi (on page 3527).

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

See the RATIONALE for *ex* for more information on *vi*. Major portions of the *vi* utility specification point to *ex* to avoid inadvertent divergence. While *ex* and *vi* have historically been implemented as a single utility, this is not required by POSIX.1-2024.

It is recognized that portions of *vi* would be difficult, if not impossible, to implement satisfactorily on a block-mode terminal, or a terminal without any form of cursor addressing, thus it is not a mandatory requirement that such features should work on all terminals. It is the intention, however, that a *vi* implementation should provide the full set of capabilities on all terminals capable of supporting them.

Historically, *vi* exited immediately if the standard input was not a terminal. POSIX.1-2024 permits, but does not require, this behavior. An end-of-file condition is not equivalent to an end-of-file character. A common end-of-file character, <control>-D, is historically a *vi* command.

The text in the STDOUT section reflects the usage of the verb *display* in this section; some implementations of *vi* use standard output to write to the terminal, but POSIX.1-2024 does not require that to be the case.

Historically, implementations reverted to open mode if the terminal was incapable of supporting full visual mode. POSIX.1-2024 requires this behavior. Historically, the open mode of *vi* behaved roughly equivalently to the visual mode, with the exception that only a single line from the edit buffer (one ``buffer line'') was kept current at any time. This line was normally displayed on the next-to-last line of a terminal with cursor addressing (and the last line performed its normal visual functions for line-oriented commands and messages). In addition, some few commands behaved differently in open mode than in visual mode. POSIX.1-2024 requires conformance to historical practice.

Historically, *ex* and *vi* implementations have expected text to proceed in the usual European/Latin order of left to right, top to bottom. There is no requirement in POSIX.1-2024 that this be the case. The specification was deliberately written using words like ``before'',

121962 ``after'', ``first'', and ``last'' in order to permit implementations to support the natural text order
121963 of the language.

121964 Historically, lines past the end of the edit buffer were marked with single <tilde> (`'~'`)
121965 characters; that is, if the one-based display was 20 lines in length, and the last line of the file was
121966 on line one, then lines 2-20 would contain only a single `'~'` character.

121967 Historically, the *vi* editor attempted to display only complete lines at the bottom of the screen (it
121968 did display partial lines at the top of the screen). If a line was too long to fit in its entirety at the
121969 bottom of the screen, the screen lines where the line would have been displayed were displayed
121970 as single `'@'` characters, instead of displaying part of the line. POSIX.1-2024 permits, but does
121971 not require, this behavior. Implementations are encouraged to attempt always to display a
121972 complete line at the bottom of the screen when doing scrolling or screen positioning by buffer
121973 lines.

121974 Historically, lines marked with `'@'` were also used to minimize output to dumb terminals over
121975 slow lines; that is, changes local to the cursor were updated, but changes to lines on the screen
121976 that were not close to the cursor were simply marked with an `'@'` sign instead of being updated
121977 to match the current text. POSIX.1-2024 permits, but does not require this feature because it is
121978 used ever less frequently as terminals become smarter and connections are faster.

121979 **Initialization in ex and vi**

121980 Historically, *vi* always had a line in the edit buffer, even if the edit buffer was ``empty''. For
121981 example:

121982 1. The *ex* command **=** executed from visual mode wrote ``1'' when the buffer was empty.

121983 2. Writes from visual mode of an empty edit buffer wrote files of a single character (a
121984 <newline>), while writes from *ex* mode of an empty edit buffer wrote empty files.

121985 3. Put and read commands into an empty edit buffer left an empty line at the top of the edit
121986 buffer.

121987 For consistency, POSIX.1-2024 does not permit any of these behaviors.

121988 Historically, *vi* did not always return the terminal to its original modes; for example, ICRNL was
121989 modified if it was not originally set. POSIX.1-2024 does not permit this behavior.

121990 **Command Descriptions in vi**

121991 Motion commands are among the most complicated aspects of *vi* to describe. With some
121992 exceptions, the text region and buffer type effect of a motion command on a *vi* command are
121993 described on a case-by-case basis. The descriptions of text regions in POSIX.1-2024 are not
121994 intended to imply direction; that is, an inclusive region from line *n* to line *n*+5 is identical to a
121995 region from line *n*+5 to line *n*. This is of more than academic interest—movements to marks can
121996 be in either direction, and, if the **wrapscan** option is set, so can movements to search points.
121997 Historically, lines are always stored into buffers in text order; that is, from the start of the edit
121998 buffer to the end. POSIX.1-2024 requires conformance to historical practice.

121999 Historically, command counts were applied to any associated motion, and were multiplicative to
122000 any supplied motion count. For example, **2cw** is the same as **c2w**, and **2c3w** is the same as **c6w**.
122001 POSIX.1-2024 requires this behavior. Historically, *vi* commands that used bigwords, words,
122002 paragraphs, and sentences as objects treated groups of empty lines, or lines that contained only
122003 <blank> characters, inconsistently. Some commands treated them as a single entity, while others
122004 treated each line separately. For example, the **w**, **W**, and **B** commands treated groups of empty
122005 lines as individual words; that is, the command would move the cursor to each new empty line.

The **e** and **E** commands treated groups of empty lines as a single word; that is, the first use would move past the group of lines. The **b** command would just beep at the user, or if done from the start of the line as a motion command, fail in unexpected ways. If the lines contained only (or ended with) <blank> characters, the **w** and **W** commands would just beep at the user, the **E** and **e** commands would treat the group as a single word, and the **B** and **b** commands would treat the lines as individual words. For consistency and simplicity of specification, POSIX.1-2024 requires that all *vi* commands treat groups of empty or blank lines as a single entity, and that movement through lines ending with <blank> characters be consistent with other movements.

Historically, *vi* documentation indicated that any number of double-quotes were skipped after punctuation marks at sentence boundaries; however, implementations only skipped single-quotes. POSIX.1-2024 requires both to be skipped.

Historically, the first and last characters in the edit buffer were word boundaries. This historical practice is required by POSIX.1-2024.

Historically, *vi* attempted to update the minimum number of columns on the screen possible, which could lead to misleading information being displayed. POSIX.1-2024 makes no requirements other than that the current character being entered is displayed correctly, leaving all other decisions in this area up to the implementation.

Historically, lines were arbitrarily folded between columns of any characters that required multiple column positions on the screen, with the exception of tabs, which terminated at the right-hand margin. POSIX.1-2024 permits the former and requires the latter. Implementations that do not arbitrarily break lines between columns of characters that occupy multiple column positions should not permit the cursor to rest on a column that does not contain any part of a character.

The historical *vi* had a problem in that all movements were by buffer lines, not by display or screen lines. This is often the right thing to do; for example, single line movements, such as **j** or **k**, should work on buffer lines. Commands like **dj**, or **j.**, where **.** is a change command, only make sense for buffer lines. It is not, however, the right thing to do for screen motion or scrolling commands like <control>-D, <control>-F, and **H**. If the window is fairly small, using buffer lines in these cases can result in completely random motion; for example, **1**<control>**-D** can result in a completely changed screen, without any overlap. This is clearly not what the user wanted. The problem is even worse in the case of the **H**, **L**, and **M** commands—as they position the cursor at the first non-<blank> of the line, they may all refer to the same location in large lines, and will result in no movement at all.

In addition, if the line is larger than the screen, using buffer lines can make it impossible to display parts of the line—there are not any commands that do not display the beginning of the line in historical *vi*, and if both the beginning and end of the line cannot be on the screen at the same time, the user suffers. Finally, the page and half-page scrolling commands historically moved to the first non-<blank> in the new line. If the line is approximately the same size as the screen, this is inadequate because the cursor before and after a <control>-D command will refer to the same location on the screen.

Implementations of *ex* and *vi* exist that do not have these problems because the relevant commands (<control>-B, <control>-D, <control>-F, <control>-U, <control>-Y, <control>-E, **H**, **L**, and **M)** operate on display (screen) lines, not (edit) buffer lines.

POSIX.1-2024 does not permit this behavior by default because the standard developers believed that users would find it too confusing. However, historical practice has been relaxed. For example, *ex* and *vi* historically attempted, albeit sometimes unsuccessfully, to never put part of a line on the last lines of a screen; for example, if a line would not fit in its entirety, no part of the line was displayed, and the screen lines corresponding to the line contained single `'@'`

characters. This behavior is permitted, but not required by POSIX.1-2024, so that it is possible for implementations to support long lines in small screens more reasonably without changing the commands to be oriented to the display (instead of oriented to the buffer). POSIX.1-2024 also permits implementations to refuse to edit any edit buffer containing a line that will not fit on the screen in its entirety.

The display area (for example, the value of the **window** edit option) has historically been ``grown'', or expanded, to display new text when local movements are done in displays where the number of lines displayed is less than the maximum possible. Expansion has historically been the first choice, when the target line is less than the maximum possible expansion value away. Scrolling has historically been the next choice, done when the target line is less than half a display away, and otherwise, the screen was redrawn. There were exceptions, however, in that *ex* commands generally always caused the screen to be redrawn. POSIX.1-2024 does not specify a standard behavior because there may be external issues, such as connection speed, the number of characters necessary to redraw as opposed to scroll, or terminal capabilities that implementations will have to accommodate.

The current line in POSIX.1-2024 maps one-to-one to a buffer line in the file. The current column does not. There are two different column values that are described by POSIX.1-2024. The first is the current column value as set by many of the *vi* commands. This value is remembered for the lifetime of the editor. The second column value is the actual position on the screen where the cursor rests. The two are not always the same. For example, when the cursor is backed by a multi-column character, the actual cursor position on the screen has historically been the last column of the character in command mode, and the first column of the character in input mode.

Commands that set the current line, but that do not set the current cursor value (for example, **j** and **k**) attempt to get as close as possible to the remembered column position, so that the cursor tends to restrict itself to a vertical column as the user moves around in the edit buffer. POSIX.1-2024 requires conformance to historical practice, requiring that the display location of the cursor on the display line be adjusted from the current column value as necessary to support this historical behavior.

Historically, only a single line (and for some terminals, a single line minus 1 column) of characters could be entered by the user for the line-oriented commands; that is, **:**, **!**, **/**, or **?**. POSIX.1-2024 permits, but does not require, this limitation.

Historically, ``soft'' errors in *vi* caused the terminal to be alerted, but no error message was displayed. As a general rule, no error message was displayed for errors in command execution in *vi*, when the error resulted from the user attempting an invalid or impossible action, or when a searched-for object was not found. Examples of soft errors included **h** at the left margin, <control>-B or **[[** at the beginning of the file, **2G** at the end of the file, and so on. In addition, errors such as **%**, **]]**, **}**, **)**, **N**, **n**, **f**, **F**, **t**, and **T** failing to find the searched-for object were soft as well. Less consistently, **/** and **?** displayed an error message if the pattern was not found, **/**, **?**, **N**, and **n** displayed an error message if no previous regular expression had been specified, and **;** did not display an error message if no previous **f**, **F**, **t**, or **T** command had occurred. Also, behavior in this area might reasonably be based on a runtime evaluation of the speed of a network connection. Finally, some implementations have provided error messages for soft errors in order to assist naive users, based on the value of a verbose edit option. POSIX.1-2024 does not list specific errors for which an error message shall be displayed. Implementations should conform to historical practice in the absence of any strong reason to diverge.

Page Backwards

The <control>-B and <control>-F commands historically considered it an error to attempt to page past the beginning or end of the file, whereas the <control>-D and <control>-U commands simply moved to the beginning or end of the file. For consistency, POSIX.1-2024 requires the latter behavior for all four commands. All four commands still consider it an error if the current line is at the beginning (<control>-B, <control>-U) or end (<control>-F, <control>-D) of the file. Historically, the <control>-B and <control>-F commands skip two lines in order to include overlapping lines when a single command is entered. This makes less sense in the presence of a *count*, as there will be, by definition, no overlapping lines. The actual calculation used by historical implementations of the *vi* editor for <control>-B was:

```
((current first line) – count x (window edit option)) +2
```

and for <control>-F was:

```
((current first line) + count x (window edit option)) –2
```

This calculation does not work well when intermixing commands with and without counts; for example, **3**<control>-F is not equivalent to entering the <control>-F command three times, and is not reversible by entering the <control>-B command three times. For consistency with other *vi* commands that take counts, POSIX.1-2024 requires a different calculation.

Scroll Forward

The 4BSD and System V implementations of *vi* differed on the initial value used by the **scroll** command. 4BSD used:

```
((window edit option) +1) /2
```

while System V used the value of the **scroll** edit option. The System V version is specified by POSIX.1-2024 because the standard developers believed that it was more intuitive and permitted the user a method of setting the scroll value initially without also setting the number of lines that are displayed.

Scroll Forward by Line

Historically, the <control>-E and <control>-Y commands considered it an error if the last and first lines, respectively, were already on the screen. POSIX.1-2024 requires conformance to historical practice. Historically, the <control>-E and <control>-Y commands had no effect in open mode. For simplicity and consistency of specification, POSIX.1-2024 requires that they behave as usual, albeit with a single line screen.

Clear and Redisplay

The historical <control>-L command refreshed the screen exactly as it was supposed to be currently displayed, replacing any `'@'` characters for lines that had been deleted but not updated on the screen with refreshed `'@'` characters. The intent of the <control>-L command is to refresh when the screen has been accidentally overwritten; for example, by a **write** command from another user, or modem noise.

122136 **Redraw Screen**

122137 The historical <control>-R command redisplayed only when necessary to update lines that had
122138 been deleted but not updated on the screen and that were flagged with `'@'` characters. There is
122139 no requirement that the screen be in any way refreshed if no lines of this form are currently
122140 displayed. POSIX.1-2024 permits implementations to extend this command to refresh lines on
122141 the screen flagged with `'@'` characters because they are too long to be displayed in the current
122142 framework; however, the current line and column need not be modified.

122143 **Search for tagstring**

122144 Historically, the first non-<blank> at or after the cursor was the first character, and all
122145 subsequent characters that were word characters, up to the end of the line, were included. For
122146 example, with the cursor on the leading <space> or on the `'#'` character in the text `"#bar@"`,
122147 the tag was `"#bar"`. On the character `'b'` it was `"bar"`, and on the `'a'` it was `"ar"`.
122148 POSIX.1-2024 requires this behavior.

122149 **Replace Text with Results from Shell Command**

122150 Historically, the **<**, **>**, and **!** commands considered most cursor motions other than line-oriented
122151 motions an error; for example, the command **>/foo<CR>** succeeded, while the command **>l**
122152 failed, even though the text region described by the two commands might be identical. For
122153 consistency, all three commands only consider entire lines and not partial lines, and the region is
122154 defined as any line that contains a character that was specified by the motion.

122155 **Move to Matching Character**

122156 Other matching characters have been left implementation-defined in order to allow extensions
122157 such as matching `'<'` and `'>'` for searching HTML, or **#ifdef**, **#else**, and **#endif** for searching C
122158 source.

122159 **Repeat Substitution**

122160 POSIX.1-2024 requires that any **c** and **g** flags specified to the previous substitute command be
122161 ignored; however, the **r** flag may still apply, if supported by the implementation.

122162 **Return to Previous (Context or Section)**

122163 The **[[**, **]]**, **(**, **)**, {, and } commands are all affected by ``section boundaries'', but in some historical
122164 implementations not all of the commands recognize the same section boundaries. This is a bug,
122165 not a feature, and a unique section-boundary algorithm was not described for each command.
122166 One special case that is preserved is that the sentence command moves to the end of the last line
122167 of the edit buffer while the other commands go to the beginning, in order to preserve the
122168 traditional character cut semantics of the sentence command. Historically, *vi* section boundaries
122169 at the beginning and end of the edit buffer were the first non-<blank> on the first and last lines
122170 of the edit buffer if one exists; otherwise, the last character of the first and last lines of the edit
122171 buffer if one exists. To increase consistency with other section locations, this has been simplified
122172 by POSIX.1-2024 to the first character of the first and last lines of the edit buffer, or the first and
122173 the last lines of the edit buffer if they are empty.

122174 Sentence boundaries were problematic in the historical *vi*. They were not only the boundaries as
122175 defined for the section and paragraph commands, but they were the first non-<blank> that
122176 occurred after those boundaries, as well. Historically, the *vi* section commands were
122177 documented as taking an optional window size as a *count* preceding the command. This was not
122178 implemented in historical versions, so POSIX.1-2024 requires that the *count* repeat the command,

for consistency with other *vi* commands.

Repeat

Historically, mapped commands other than text input commands could not be repeated using the **period** command. POSIX.1-2024 requires conformance to historical practice.

The restrictions on the interpretation of special characters (for example, <control>-H) in the repetition of text input mode commands is intended to match historical practice. For example, given the input sequence:

```
iab<control>-H<control>-H<control>-Hdef<escape>
```

the user should be informed of an error when the sequence is first entered, but not during a command repetition. The character <control>-T is specifically exempted from this restriction. Historical implementations of *vi* ignored <control>-T characters that were input in the original command during command repetition. POSIX.1-2024 prohibits this behavior.

Find Regular Expression

Historically, commands did not affect the line searched to or from if the motion command was a search (**/**, **?**, **N**, **n**) and the final position was the start/end of the line. There were some special cases and *vi* was not consistent. POSIX.1-2024 does not permit this behavior, for consistency. Historical implementations permitted but were unable to handle searches as motion commands that wrapped (that is, due to the edit option **wrapscan**) to the original location. POSIX.1-2024 requires that this behavior be treated as an error.

Historically, the syntax `"/RE/0"` was used to force the command to cut text in line mode. POSIX.1-2024 requires conformance to historical practice.

Historically, in open mode, a **z** specified to a search command redisplayed the current line instead of displaying the current screen with the current line highlighted. For consistency and simplicity of specification, POSIX.1-2024 does not permit this behavior.

Historically, trailing **z** commands were permitted and ignored if entered as part of a search used as a motion command. For consistency and simplicity of specification, POSIX.1-2024 does not permit this behavior.

Execute an ex Command

Historically, *vi* implementations restricted the commands that could be entered on the colon command line (for example, **append** and **change**), and some other commands were known to cause them to fail catastrophically. For consistency, POSIX.1-2024 does not permit these restrictions. When executing an *ex* command by entering **:**, it is not possible to enter a <newline> as part of the command because it is considered the end of the command. A different approach is to enter *ex* command mode by using the *vi* **Q** command (and later resuming visual mode with the *ex* **vi** command). In *ex* command mode, the single-line limitation does not exist. So, for example, the following is valid:

```
Q
s/break here/break\
here/
vi
```

POSIX.1-2024 requires that, if the *ex* command overwrites any part of the screen that would be erased by a refresh, *vi* pauses for a character from the user. Historically, this character could be any character; for example, a character input by the user before the message appeared, or even a

mapped character. This is probably a bug, but implementations that have tried to be more rigorous by requiring that the user enter a specific character, or that the user enter a character after the message was displayed, have been forced by user indignation back into historical behavior. POSIX.1-2024 requires conformance to historical practice.

Shift Left (Right)

Refer to the Rationale for the **!** and **/** commands. Historically, the **<** and **>** commands sometimes moved the cursor to the first non-<blank> (for example if the command was repeated or with _ as the motion command), and sometimes left it unchanged. POSIX.1-2024 does not permit this inconsistency, requiring instead that the cursor always move to the first non-<blank>. Historically, the **<** and **>** commands did not support buffer arguments, although some implementations allow the specification of an optional buffer. This behavior is neither required nor disallowed by POSIX.1-2024.

Execute

Historically, buffers could execute other buffers, and loops, infinite and otherwise, were possible. POSIX.1-2024 requires conformance to historical practice. The **buffer* syntax of *ex* is not required in *vi*, because it is not historical practice and has been used in some *vi* implementations to support additional scripting languages.

Reverse Case

Historically, the ~ command ignored any associated *count*, and acted only on the characters in the current line. For consistency with other *vi* commands, POSIX.1-2024 requires that an associated *count* act on the next *count* characters, and that the command move to subsequent lines if warranted by *count*, to make it possible to modify large pieces of text in a reasonably efficient manner. There exist *vi* implementations that optionally require an associated motion command for the ~ command. Implementations supporting this functionality are encouraged to base it on the **tildedop** edit option and handle the text regions and cursor positioning identically to the **yank** command.

Append

Historically, *count*s specified to the **A**, **a**, **I**, and **i** commands repeated the input of the first line *count* times, and did not repeat the subsequent lines of the input text. POSIX.1-2024 requires that the entire text input be repeated *count* times.

Move Backward to Preceding Word

Historically, *vi* became confused if word commands were used as motion commands in empty files. POSIX.1-2024 requires that this be an error. Historical implementations of *vi* had a large number of bugs in the word movement commands, and they varied greatly in behavior in the presence of empty lines, ``words'' made up of a single character, and lines containing only <blank> characters. For consistency and simplicity of specification, POSIX.1-2024 does not permit this behavior.

122259 **Change to End-of-Line**

122260 Some historical implementations of the **C** command did not behave as described by
122261 POSIX.1-2024 when the **$** key was remapped because they were implemented by pushing the **$**
122262 key onto the input queue and reprocessing it. POSIX.1-2024 does not permit this behavior.
122263 Historically, the **C**, **S**, and **s** commands did not copy replaced text into the numeric buffers. For
122264 consistency and simplicity of specification, POSIX.1-2024 requires that they behave like their
122265 respective **c** commands in all respects.

122266 **Delete**

122267 Historically, lines in open mode that were deleted were scrolled up, and an @ glyph written over
122268 the beginning of the line. In the case of terminals that are incapable of the necessary cursor
122269 motions, the editor erased the deleted line from the screen. POSIX.1-2024 requires conformance
122270 to historical practice; that is, if the terminal cannot display the `'@'` character, the line cannot
122271 remain on the screen.

122272 **Delete to End-of-Line**

122273 Some historical implementations of the **D** command did not behave as described by
122274 POSIX.1-2024 when the **$** key was remapped because they were implemented by pushing the **$**
122275 key onto the input queue and reprocessing it. POSIX.1-2024 does not permit this behavior.

122276 **Join**

122277 An historical oddity of *vi* is that the commands **J**, **1J**, and **2J** are all equivalent. POSIX.1-2024
122278 requires conformance to historical practice. The *vi* **J** command is specified in terms of the *ex* **join**
122279 command with an *ex* command *count* value. The address correction for a *count* that is past the
122280 end of the edit buffer is necessary for historical compatibility for both *ex* and *vi*.

122281 **Mark Position**

122282 Historical practice is that only lowercase letters, plus backquote and single-quote, could be used
122283 to mark a cursor position. POSIX.1-2024 requires conformance to historical practice, but
122284 encourages implementations to support other characters as marks as well.

122285 **Repeat Regular Expression Find (Forward and Reverse)**

122286 Historically, the **N** and **n** commands could not be used as motion components for the **c**
122287 command. With the exception of the **cN** command, which worked if the search crossed a line
122288 boundary, the text region would be discarded, and the user would not be in text input mode. For
122289 consistency and simplicity of specification, POSIX.1-2024 does not permit this behavior.

122290 **Insert Empty Line (Below and Above)**

122291 Historically, counts to the **O** and **o** commands were used as the number of physical lines to
122292 open, if the terminal was dumb and the **slowopen** option was not set. This was intended to
122293 minimize traffic over slow connections and repainting for dumb terminals. POSIX.1-2024 does
122294 not permit this behavior, requiring that a *count* to the open command behave as for other text
122295 input commands. This change to historical practice was made for consistency, and because a
122296 superset of the functionality is provided by the **slowopen** edit option.

Put from Buffer (Following and Before)

Historically, *count*s to the **p** and **P** commands were ignored if the buffer was a line mode buffer, but were (mostly) implemented as described in POSIX.1-2024 if the buffer was a character mode buffer. Because implementations exist that do not have this limitation, and because pasting lines multiple times is generally useful, POSIX.1-2024 requires that *count* be supported for all **p** and **P** commands.

Historical implementations of *vi* were widely known to have major problems in the **p** and **P** commands, particularly when unusual regions of text were copied into the edit buffer. The standard developers viewed these as bugs, and they are not permitted for consistency and simplicity of specification.

Historically, a **P** or **p** command (or an *ex* **put** command executed from open or visual mode) executed in an empty file, left an empty line as the first line of the file. For consistency and simplicity of specification, POSIX.1-2024 does not permit this behavior.

Replace Character

Historically, the **r** command did not correctly handle the *erase* and *word erase* characters as arguments, nor did it handle an associated *count* greater than 1 with a <carriage-return> argument, for which it replaced *count* characters with a single <newline>. POSIX.1-2024 does not permit these inconsistencies.

Historically, the **r** command permitted the <control>-V escaping of entered characters, such as <ESC> and the <carriage-return>; however, it required two leading <control>-V characters instead of one. POSIX.1-2024 requires that this be changed for consistency with the other text input commands of *vi*.

Historically, it is an error to enter the **r** command if there are less than *count* characters at or after the cursor in the line. While a reasonable and unambiguous extension would be to permit the **r** command on empty lines, it would require that too large a *count* be adjusted to match the number of characters at or after the cursor for consistency, which is sufficiently different from historical practice to be avoided. POSIX.1-2024 requires conformance to historical practice.

Replace Characters

Historically, if there were **autoindent** characters in the line on which the **R** command was run, and **autoindent** was set, the first <newline> would be properly indented and no characters would be replaced by the <newline>. Each additional <newline> would replace *n* characters, where *n* was the number of characters that were needed to indent the rest of the line to the proper indentation level. This behavior is a bug and is not permitted by POSIX.1-2024.

Undo

Historical practice for cursor positioning after undoing commands was mixed. In most cases, when undoing commands that affected a single line, the cursor was moved to the start of added or changed text, or immediately after deleted text. However, if the user had moved from the line being changed, the column was either set to the first non-<blank>, returned to the origin of the command, or remained unchanged. When undoing commands that affected multiple lines or entire lines, the cursor was moved to the first character in the first line restored. As an example of how inconsistent this was, a search, followed by an **o** text input command, followed by an **undo** would return the cursor to the location where the **o** command was entered, but a **cw** command followed by an **o** command followed by an **undo** would return the cursor to the first non-<blank> of the line. POSIX.1-2024 requires the most useful of these behaviors, and discards the least useful, in the interest of consistency and simplicity of specification.

122342 **Yank**

122343 Historically, the **yank** command did not move to the end of the motion if the motion was in the
122344 forward direction. It moved to the end of the motion if the motion was in the backward
122345 direction, except for the _ command, or for the **G** and ' commands when the end of the motion
122346 was on the current line. This was further complicated by the fact that for a number of motion
122347 commands, the **yank** command moved the cursor but did not update the screen; for example, a
122348 subsequent command would move the cursor from the end of the motion, even though the
122349 cursor on the screen had not reflected the cursor movement for the **yank** command.
122350 POSIX.1-2024 requires that all **yank** commands associated with backward motions move the
122351 cursor to the end of the motion for consistency, and specifically, to make ' commands as motions
122352 consistent with search patterns as motions.

122353 **Yank Current Line**

122354 Some historical implementations of the **Y** command did not behave as described by
122355 POSIX.1-2024 when the '_' key was remapped because they were implemented by pushing the
122356 '_' key onto the input queue and reprocessing it. POSIX.1-2024 does not permit this behavior.

122357 **Redraw Window**

122358 Historically, the **z** command always redrew the screen. This is permitted but not required by
122359 POSIX.1-2024, because of the frequent use of the **z** command in macros such as **map n nz.** for
122360 screen positioning, instead of its use to change the screen size. The standard developers
122361 believed that expanding or scrolling the screen offered a better interface for users. The ability to
122362 redraw the screen is preserved if the optional new window size is specified, and in the
122363 <control>-L and <control>-R commands.

122364 The semantics of **z^** are confusing at best. Historical practice is that the screen before the screen
122365 that ended with the specified line is displayed. POSIX.1-2024 requires conformance to historical
122366 practice.

122367 Historically, the **z** command would not display a partial line at the top or bottom of the screen. If
122368 the partial line would normally have been displayed at the bottom of the screen, the command
122369 worked, but the partial line was replaced with '@' characters. If the partial line would normally
122370 have been displayed at the top of the screen, the command would fail. For consistency and
122371 simplicity of specification, POSIX.1-2024 does not permit this behavior.

122372 Historically, the **z** command with a line specification of 1 ignored the command. For consistency
122373 and simplicity of specification, POSIX.1-2024 does not permit this behavior.

122374 Historically, the **z** command did not set the cursor column to the first non-<blank> for the
122375 character if the first screen was to be displayed, and was already displayed. For consistency and
122376 simplicity of specification, POSIX.1-2024 does not permit this behavior.

122377 **Input Mode Commands in vi**

122378 Historical implementations of *vi* did not permit the user to erase more than a single line of input,
122379 or to use normal erase characters such as *line erase*, *worderase*, and *erase* to erase **autoindent**
122380 characters. As there exist implementations of *vi* that do not have these limitations, both
122381 behaviors are permitted, but only historical practice is required. In the case of these extensions,
122382 *vi* is required to pause at the **autoindent** and previous line boundaries.

122383 Historical implementations of *vi* updated only the portion of the screen where the current cursor
122384 character was displayed. For example, consider the *vi* input keystrokes:

122385 `iabcd<escape>0C<tab>`

Historically, the <tab> would overwrite the characters `"abcd"` when it was displayed. Other implementations replace only the `'a'` character with the <tab>, and then push the rest of the characters ahead of the cursor. Both implementations have problems. The historical implementation is probably visually nicer for the above example; however, for the keystrokes:

```
iabcd<ESC>0R<tab><ESC>
```

the historical implementation results in the string `"bcd"` disappearing and then magically reappearing when the <ESC> character is entered. POSIX.1-2024 requires the former behavior when overwriting erase-columns—that is, overwriting characters that are no longer logically part of the edit buffer—and the latter behavior otherwise.

Historical implementations of *vi* discarded the <control>-D and <control>-T characters when they were entered at places where their command functionality was not appropriate. POSIX.1-2024 requires that the <control>-T functionality always be available, and that <control>-D be treated as any other key when not operating on **autoindent** characters.

NUL

Some historical implementations of *vi* limited the number of characters entered using the NUL input character to 256 bytes. POSIX.1-2024 permits this limitation; however, implementations are encouraged to remove this limit.

<control>-D

See also Rationale for the input mode command <newline>. The hidden assumptions in the <control>-D command (and in the *vi* **autoindent** specification in general) is that <space> characters take up a single column on the screen and that <tab> characters are comprised of an integral number of <space> characters.

<newline>

Implementations are permitted to rewrite **autoindent** characters in the line when <newline>, <carriage-return>, <control>-D, and <control>-T are entered, or when the **shift** commands are used, because historical implementations have both done so and found it necessary to do so. For example, a <control>-D when the cursor is preceded by a single <tab>, with **tabstop** set to 8, and **shiftwidth** set to 3, will result in the <tab> being replaced by several <space> characters.

<control>-T

See also the Rationale for the input mode command <newline>. Historically, <control>-T only worked if no non-<blank> characters had yet been input in the current input line. In addition, the characters inserted by <control>-T were treated as **autoindent** characters, and could not be erased using normal user erase characters. Because implementations exist that do not have these limitations, and as moving to a column boundary is generally useful, POSIX.1-2024 requires that both limitations be removed.

<control>-V

Historically, *vi* used **^V**, regardless of the value of the literal-next character of the terminal. POSIX.1-2024 requires conformance to historical practice.

The uses described for <control>-V can also be accomplished with <control>-Q, which is useful on terminals that use <control>-V for the down-arrow function. However, most historical implementations use <control>-Q for the *termios* START character, so the editor will generally not receive the <control>-Q unless **stty ixon** mode is set to off. (In addition, some historical implementations of *vi* explicitly set **ixon** mode to on, so it was difficult for the user to set it to off.) Any of the command characters described in POSIX.1-2024 can be made ineffective by their selection as *termios* control characters, using the *stty* utility or other methods described in the System Interfaces volume of POSIX.1-2024.

<ESC>

Historically, SIGINT alerted the terminal when used to end input mode. This behavior is permitted, but not required, by POSIX.1-2024.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

ed, *ex*, *stty*

XBD Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The APPLICATION USAGE section is added.

The obsolescent SYNOPSIS is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **reindent** command description is added.

The *vi* utility has been extensively rewritten for alignment with the IEEE P1003.2b draft standard.

IEEE PASC Interpretations 1003.2 #57, #62, #63, #64, #78, and #188 are applied.

IEEE PASC Interpretation 1003.2 #207 is applied, clarifying the description of the **R** command in a manner similar to the descriptions of other text input mode commands such as **i**, **o**, and **O**.

The **–l** option is removed.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/41 is applied, adding [*count*] to the Synopsis for **[[**.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/42 is applied, adding [*count*] to the

Synopsis for]].

Issue 7

Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that `'+'` may be recognized as an option delimiter in the OPTIONS section.

Austin Group Interpretation 1003.1-2001 #087 is applied, updating the Put from Buffer Before (**P**) command description to address multi-line requirements.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0202 [812] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1310 is applied, changing the CONSEQUENCES OF ERRORS section.

Austin Group Defect 1676 is applied, changing the typeface of some bold text in the RATIONALE section.

NAME

wait — await process completion

SYNOPSIS

```
wait [pid...]
```

DESCRIPTION

The *wait* utility shall wait for one or more child processes whose process IDs are known in the current shell execution environment (see Section 2.13, on page 2522) to terminate.

If the *wait* utility is invoked with no operands, it shall wait until all process IDs known to the invoking shell have terminated and exit with a zero exit status.

If one or more *pid* operands are specified that represent known process IDs, the *wait* utility shall wait until all of them have terminated. If one or more *pid* operands are specified that represent unknown process IDs, *wait* shall treat them as if they were known process IDs that exited with exit status 127. The exit status returned by the *wait* utility shall be the exit status of the process requested by the last *pid* operand.

Once a process ID that is known in the current shell execution environment (see Section 2.13, on page 2522) has been successfully waited for, it shall be removed from the list of process IDs that are known in the current shell execution environment. If the process ID is associated with a background job, the corresponding job shall also be removed from the list of background jobs.

OPTIONS

None.

OPERANDS

The following operand shall be supported:

pid One of the following:

1. The unsigned decimal integer process ID of a child process whose termination the utility is to wait for.
2. A job ID (see XBD Section 3.182, on page 57) that identifies a process group in the case of a job-control background job, or a process ID in the case of a non-job-control background job (if supported), to be waited for. The job ID notation is applicable only for invocations of *wait* in the current shell execution environment; see Section 2.13 (on page 2522). The exit status of *wait* shall be determined by the exit status of the last pipeline to be executed.

   Note: The job ID type of *pid* is only available on systems supporting the User Portability Utilities option or supporting non-job-control background jobs.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *wait*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

122519 *LC_ALL* If set to a non-empty string value, override the values of all the other
122520 internationalization variables.

122521 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
122522 characters (for example, single-byte as opposed to multi-byte characters in
122523 arguments).

122524 *LC_MESSAGES*
122525 Determine the locale that should be used to affect the format and contents of
122526 diagnostic messages written to standard error.

122527 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

122528 **ASYNCHRONOUS EVENTS**
122529 Default.

122530 **STDOUT**
122531 Not used.

122532 **STDERR**
122533 The standard error shall be used only for diagnostic messages.

122534 **OUTPUT FILES**
122535 None.

122536 **EXTENDED DESCRIPTION**
122537 None.

122538 **EXIT STATUS**
122539 If one or more operands were specified, all of them have terminated or were not known in the
122540 invoking shell execution environment, and the status of the last operand specified is known,
122541 then the exit status of *wait* shall be the status of the last operand specified. If the process
122542 terminated abnormally due to the receipt of a signal, the exit status shall be greater than 128 and
122543 shall be distinct from the exit status generated by other signals, but the exact value is
122544 unspecified. (See the *kill* **–l** option.) Otherwise, the *wait* utility shall exit with one of the
122545 following values:

122546 0 The *wait* utility was invoked with no operands and all process IDs known by the
122547 invoking shell have terminated.

122548 1-126 The *wait* utility detected an error.

122549 127 The process ID specified by the last *pid* operand specified is not known in the invoking
122550 shell execution environment.

122551 **CONSEQUENCES OF ERRORS**
122552 Default.

122553 **APPLICATION USAGE**
122554 This utility is required to be intrinsic. See Section 1.7 (on page 2470) for details.

122555 On most implementations, *wait* is a shell built-in. If it is called in a subshell or separate utility
122556 execution environment, such as one of the following:

```
122557 (wait)
122558 nohup wait ...
122559 find . -exec wait ... \;
```

122560 it returns immediately because there are no known process IDs to wait for in those
122561 environments.

The use of job ID notation is not dependent on job control being enabled. When job control has been disabled using *set* **+m**, *wait* can still be used to wait for the process group associated with a job-control background job, or the process ID associated with a non-control background job (if supported), using

```
wait %<background job number>
```

See also the RATIONALE for *jobs* and *kill*.

The shell is allowed to discard the status of any process if it determines that the application cannot get the process ID for that process from the shell. It is also required to remember only {CHILD_MAX} number of processes in this way. Since the only way to get the process ID from the shell is by using the `'!'` shell parameter, the shell is allowed to discard the status of an asynchronous AND-OR list if `"$!"` was not referenced before another asynchronous AND-OR list was started. (This means that the shell only has to keep the status of the last asynchronous AND-OR list started if the application did not reference `"$!"`. If the implementation of the shell is smart enough to determine that a reference to `"$!"` was not saved anywhere that the application can retrieve it later, it can use this information to trim the list of saved information. Note also that a successful call to *wait* with no operands discards the exit status of all asynchronous AND-OR lists.)

If the exit status of *wait* is greater than 128, there is no way for the application to know if the waited-for process exited with that value or was killed by a signal. Since most utilities exit with small values, there is seldom any ambiguity. Even in the ambiguous cases, most applications just need to know that the asynchronous job failed; it does not matter whether it detected an error and failed or was killed and did not complete its job normally.

Some historical shells returned from wait when a process stops instead of only when it terminates. This standard does not allow wait to return when a process stops for two reasons:

1. The vast majority, if not all, shell scripts that use *wait* (without using an extension) expect it not to return until the process terminates.
2. It is not possible to write a portable shell script that can correctly handle *wait* returning when a process stops, because an exit status indicating a process was stopped by a signal cannot be distinguished from one indicating that the process called *exit*( ) with the same value.

The standard developers considered allowing interactive shells to return from *wait* when a process stops, since the interactive user would see a message which would allow them to tell whether the process stopped or terminated. However, they decided that it would be inadvisable to introduce an inconsistency between interactive and non-interactive shells, particularly as the most likely use of *wait* in an interactive shell is to try out commands before putting them in a shell script. Implementations can provide an extension that could be used to request that *wait* returns when a process stops. It is recommended that any such extension uses a different method of returning information about the wait status of the process so that the information can be unambiguous. One suitable method would be an option that takes a variable name as an option-argument. The named variable would be set to a numeric value and the exit status of wait would indicate whether this value is an exit value or a signal number, and whether the signal terminated the process or stopped it. Such an extension would also provide a way for shell scripts to obtain the full exit value (as would be returned by *waitid*( )).

EXAMPLES

Although the exact value used when a process is terminated by a signal is unspecified, if it is known that a signal terminated a process, a script can still reliably determine which signal by using *kill* as shown by the following script:


```
sleep 1000&
pid=$!
kill -kill $pid
wait $pid
echo $pid was terminated by a SIG$(kill -l $?) signal.
```

If the following sequence of commands is run in less than 31 seconds:

```
sleep 257 | sleep 31 &
jobs -l %%
```

either of the following commands returns the exit status of the second *sleep* in the pipeline:

```
wait <pid of sleep 31>
wait %%
```

RATIONALE

The description of *wait* does not refer to the *waitpid*() function from the System Interfaces volume of POSIX.1-2024 because that would needlessly overspecify this interface. However, the wording means that *wait* is required to wait for an explicit process when it is given an argument so that the status information of other processes is not consumed. Historical implementations use the *wait*() function defined in the System Interfaces volume of POSIX.1-2024 until *wait*() returns the requested process ID or finds that the requested process does not exist. Because this means that a shell script could not reliably get the status of all background children if a second background job was ever started before the first job finished, it is recommended that the *wait* utility use a method such as the functionality provided by the *waitpid*() function.

The ability to wait for multiple *pid* operands was adopted from the KornShell.

This new functionality was added because it is needed to determine the exit status of any asynchronous AND-OR list accurately. The only compatibility problem that this change creates is for a script like

```
while sleep 60 do
    job& echo Job started $(date) as $!  done
```

which causes the shell to monitor all of the jobs started until the script terminates or runs out of memory. This would not be a problem if the loop did not reference `"$!"` or if the script would occasionally *wait* for jobs it started.

FUTURE DIRECTIONS

A future version of this standard may add an option which takes a variable name as an option-argument, allowing *wait* to return information about the wait status of a process in an unambiguous way.

SEE ALSO

Chapter 2 (on page 2472), *kill*, *sh*

XBD Section 3.182 (on page 57), Chapter 8 (on page 167)

XSH *wait*()

CHANGE HISTORY

First released in Issue 2.

Issue 8

Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that this utility is required to be intrinsic.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

122653 Austin Group Defect 1254 is applied, updating various requirements for the *jobs* utility to
122654 account for the addition of Section 2.11 (on page 2518).

NAME

wc — word, line, and byte or character count

SYNOPSIS

```
wc [-c|-m] [-lw] [file...]
```

DESCRIPTION

The *wc* utility shall read one or more input files and, by default, write the number of <newline> characters, words, and bytes contained in each input file to the standard output.

The utility also shall write a total count for all named files, if more than one input file is specified.

The *wc* utility shall consider a *word* to be a non-zero-length string of characters delimited by white space.

OPTIONS

The *wc* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

−c Write to the standard output the number of bytes in each input file.

−l Write to the standard output the number of <newline> characters in each input file.

−m Write to the standard output the number of characters in each input file.

−w Write to the standard output the number of words in each input file.

When any option is specified, *wc* shall report only the information requested by the specified options.

OPERANDS

The following operand shall be supported:

file A pathname of an input file. If no *file* operands are specified, the standard input shall be used.

STDIN

The standard input shall be used if no *file* operands are specified, and shall be used if a *file* operand is `'-'` and the implementation treats the `'-'` as meaning standard input. Otherwise, the standard input shall not be used. See the INPUT FILES section.

INPUT FILES

The input files may be of any type.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *wc*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and which characters are defined as white-space characters.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS
Default.

STDOUT
By default, the standard output shall contain an entry for each input file of the form:

`"%d %d %d %s\n", <newlines>, <words>, <bytes>, <file>`

If the **–m** option is specified, the number of characters shall replace the **<***bytes***>** field in this format.

If any options are specified and the **–l** option is not specified, the number of **<**newline**>** characters shall not be written.

If any options are specified and the **–w** option is not specified, the number of words shall not be written.

If any options are specified and neither **–c** nor **–m** is specified, the number of bytes or characters shall not be written.

If no input *file* operands are specified, no name shall be written and no **<**blank**>** characters preceding the pathname shall be written.

If more than one input *file* operand is specified, an additional line shall be written, of the same format as the other lines, except that the word **total** (in the POSIX locale) shall be written instead of a pathname and the total of each column shall be written as appropriate. Such an additional line, if any, is written at the end of the output.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE

The **−m** option is not a switch, but an option at the same level as **−c**. Thus, to produce the full default output with character counts instead of bytes, the command required is:

```
wc -mlw
```

EXAMPLES

None.

RATIONALE

The output file format pseudo-*printf*( ) string differs from the System V version of *wc*:

```
"%7d%7d%7d %s\n"
```

which produces possibly ambiguous and unparsable results for very large files, as it assumes no number shall exceed six digits.

Some historical implementations use only <space>, <tab>, and <newline> as word separators. The equivalent of the ISO C standard *isspace*( ) function is more appropriate.

The **−c** option stands for ``character'' count, even though it counts bytes. This stems from the sometimes erroneous historical view that bytes and characters are the same size. Due to international requirements, the **−m** option (reminiscent of ``multi-byte'') was added to obtain actual character counts.

Early proposals only specified the results when input files were text files. The current specification more closely matches historical practice. (Bytes, words, and <newline> characters are counted separately and the results are written when an end-of-file is detected.)

Historical implementations of the *wc* utility only accepted one argument to specify the options **−c**, **−l**, and **−w**. Some of them also had multiple occurrences of an option cause the corresponding count to be written multiple times and had the order of specification of the options affect the order of the fields on output, but did not document either of these. Because common usage either specifies no options or only one option, and because none of this was documented, the changes required by this volume of POSIX.1-2024 should not break many historical applications (and do not break any historical conforming applications).

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

cksum

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 7

Austin Group Interpretation 1003.1-2001 #092 is applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

what — identify SCCS files (**DEVELOPMENT**)

SYNOPSIS

XSI

```
what [-s] file...
```

DESCRIPTION

The *what* utility shall search the given files for all occurrences of the pattern that *get* (see *get*) substitutes for the %**Z**% keyword (`"@(#)"`). The *what* utility shall write to standard output the identification string that follows up to, but not including, the first occurrence of one of the following: <double-quote> (`'"'`), <greater-than-sign> (`'>'`), <newline>, <backslash> (`'\\'`), <NUL> (`'\0'`), or an end-of-file condition on the input file. If not at end-of-file, the *what* utility shall then look for the next occurrence of `"@(#)"` after one of those characters.

OPTIONS

The *what* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

−s Write at most one identification string for each file. After locating and writing to standard output the identification string following the first pattern (if any) in a file, no further data shall be read from that file and the search shall recommence from the beginning of the next file, if any.

OPERANDS

The following operands shall be supported:

file A pathname of a file to search.

STDIN

Not used.

INPUT FILES

The input files shall be of any file type.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *what*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

Default.

122820 **STDOUT**

122821 For each *file* operand, the standard output shall consist of:

122822 `"%s:\n", <pathname>`

122823 followed by zero or more of:

122824 `"\t%s\n", <identification string>`

122825 one for each identification string located.

122826 **STDERR**

122827 The standard error shall be used only for diagnostic messages.

122828 **OUTPUT FILES**

122829 None.

122830 **EXTENDED DESCRIPTION**

122831 None.

122832 **EXIT STATUS**

122833 The following exit values shall be returned:

122834 0 One or more matches were found and the output specified in STDOUT was successfully
122835 written to standard output.

122836 1 No matches were found or an error occurred.

122837 **CONSEQUENCES OF ERRORS**

122838 Default.

122839 **APPLICATION USAGE**

122840 The *what* utility is intended to be used in conjunction with the SCCS command *get*, which
122841 automatically inserts identifying information, but it can also be used where the information is
122842 inserted by any other means.

122843 When the string `"@(#)"` is included in a library routine in a shared library, it might not be found
122844 in an **a.out** file using that library routine.

122845 **EXAMPLES**

122846 If the C-language program in file **f.c** contains:

122847
```
char ident[] = "@(#)identification information";
```

122848 and **f.c** is compiled to yield **f.o** and **a.out**, then the command:

122849
```
what f.c f.o a.out
```

122850 writes:

122851
```
f.c:
    identification information
    ...
f.o:
    identification information
    ...
a.out:
    identification information
    ...
```


RATIONALE

This standard requires that when the **–s** option is used, *what* does not continue reading from the current file after writing the first identification string. This might seem an unimportant detail, but applications would experience different behavior if a *file* operand named a FIFO special file and *what* waited for an end-of-file condition rather than closing the file straight away.

FUTURE DIRECTIONS

If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

get

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1512 is applied, changing the EXIT STATUS section.

Austin Group Defect 1538 is applied, clarifying the **–s** option.

Austin Group Defect 1563 is applied, clarifying the output format when the *what* utility finds multiple identification strings in one file.

NAME

who — display who is on the system

SYNOPSIS

```
who [-mTu] [-abdHlprt] [file]
who [-mu] -s [-bHlprt] [file]
who -q [file]
who am i
who am I
```

DESCRIPTION

The *who* utility shall list various pieces of information about accessible users. The domain of accessibility is implementation-defined.

Based on the options given, *who* can also list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process ID of the command interpreter for each current system user.

OPTIONS

The *who* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported. The metavariables, such as <*line*>, refer to fields described in the STDOUT section.

–a Process the implementation-defined database or named file with the **–b**, **–d**, **–l**, **–p**, **–r**, **–t**, **–T** and **–u** options turned on.

–b Write the time and date of the last system reboot. The system reboot time is the time at which the implementation is able to commence running processes.

–d Write a list of all processes that have expired and not been respawned by the *init* system process. The <*exit*> field shall appear for dead processes and contain the termination and exit values of the dead process. This can be useful in determining why a process terminated.

–H Write column headings above the regular output.

–l (The letter ell.) List only those lines on which the system is waiting for someone to login. The <*name*> field shall be **LOGIN** in such cases. Other fields shall be the same as for user entries except that the <*state*> field does not exist.

–m Output only information about the current terminal.

–p List any other process that is currently active and has been previously spawned by *init*.

–q (Quick.) List only the names and the number of users currently logged on. When this option is used, all other options shall be ignored.

–r Write the current *run-level* of the *init* process.

–s List only the <*name*>, <*line*>, and <*time*> fields. This is the default case.

–t Indicate the last change to the system clock.

–T Show the state of each terminal, as described in the STDOUT section.

122924 **–u** Write ``idle time'' for each displayed user in addition to any other information. The
122925 idle time is the time since any activity occurred on the user's terminal. The method
122926 XSI of determining this is unspecified. This option shall list only those users who are
122927 currently logged in. The <*name*> is the user's login name. The <*line*> is the name
122928 of the line as found in the directory **/dev**. The <*time*> is the time that the user
122929 logged in. The <*activity*> is the number of hours and minutes since activity last
122930 occurred on that particular line. A dot indicates that the terminal has seen activity
122931 in the last minute and is therefore ``current''. If more than twenty-four hours have
122932 elapsed or the line has not been used since boot time, the entry shall be marked
122933 <*old*>. This field is useful when trying to determine whether a person is working at
122934 the terminal or not. The <*pid*> is the process ID of the user's login process.

122935 **OPERANDS**

122936 XSI The following operands shall be supported:

122937 **am i**, **am I** In the POSIX locale, limit the output to describing the invoking user, equivalent to
122938 the **–m** option. The **am** and **i** or **I** need to be separate arguments.

122939 *file* Specify a pathname of a file to substitute for the implementation-defined database
122940 of logged-on users that *who* uses by default.

122941 **STDIN**

122942 Not used.

122943 **INPUT FILES**

122944 None.

122945 **ENVIRONMENT VARIABLES**

122946 The following environment variables shall affect the execution of *who*:

122947 *LANG* Provide a default value for the internationalization variables that are unset or null.
122948 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
122949 variables used to determine the values of locale categories.)

122950 *LC_ALL* If set to a non-empty string value, override the values of all the other
122951 internationalization variables.

122952 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
122953 characters (for example, single-byte as opposed to multi-byte characters in
122954 arguments).

122955 *LC_MESSAGES*
122956 Determine the locale that should be used to affect the format and contents of
122957 diagnostic messages written to standard error.

122958 *LC_TIME* Determine the locale used for the format and contents of the date and time strings.

122959 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

122960 *TZ* Determine the timezone used when writing date and time information. If *TZ* is
122961 unset or null, an unspecified default timezone shall be used.

122962 **ASYNCHRONOUS EVENTS**

122963 Default.

122964 **STDOUT**

122965 The *who* utility shall write its default format to the standard output in an implementation-
122966 defined format, subject only to the requirement of containing the information described above.

XSI OF XSI-conformant systems shall write the default information to the standard output in the following general format:

`<name>[<state>]<line><time>[<activity>][<pid>][<comment>][<exit>]`

For the –**b** option, **<***line***>** shall be `"system boot"`. The **<***name***>** is unspecified.

The following format shall be used for the –**T** option:

```
"%s %c %s %s\n" <name>, <terminal state>, <terminal name>,
    <time of login>
```

where **<***terminal state***>** is one of the following characters:

`+` The terminal allows write access to other users.

`–` The terminal denies write access to other users.

`?` The terminal write-access state cannot be determined.

`<space>` This entry is not associated with a terminal.

In the POSIX locale, the **<***time of login***>** shall be equivalent in format to the output of:

```
date +"%b %e %H:%M"
```

If the –**u** option is used with –**T**, the idle time shall be added to the end of the previous format in an unspecified format.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The name *init* used for the system process is the most commonly used on historical systems, but it may vary.

The ``domain of accessibility'' referred to is a broad concept that permits interpretation either on a very secure basis or even to allow a network-wide implementation like the historical *rwho*.

EXAMPLES

None.

RATIONALE

Due to differences between historical implementations, the base options provided were a compromise to allow users to work with those functions. The standard developers also considered removing all the options, but felt that these options offered users valuable functionality. Additional options to match historical systems are available on XSI-conformant

systems.

It is recognized that the *who* command may be of limited usefulness, especially in a multi-level secure environment. The standard developers considered, however, that having some standard method of determining the ``accessibility'' of other users would aid user portability.

No format was specified for the default *who* output for systems not supporting the XSI option. In such a user-oriented command, designed only for human use, this was not considered to be a deficiency.

The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, and *write* require that they use the same format.

It is acceptable for an implementation to produce no output for an invocation of *who* **mil**.

FUTURE DIRECTIONS

None.

SEE ALSO

mesg

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

Issue 7

SD5-XCU-ERN-58 is applied, clarifying the –**b** option.

The *who* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

write — write to another user

SYNOPSIS

```
write user_name [terminal]
```

DESCRIPTION

The *write* utility shall read lines from the standard input and write them to the terminal of the specified user. When first invoked, it shall write the message:

```
Message from sender-login-id (sending-terminal) [date]...
```

to *user_name*. When it has successfully completed the connection, the sender's terminal shall be alerted twice to indicate that what the sender is typing is being written to the recipient's terminal.

If the recipient wants to reply, this can be accomplished by typing:

```
write sender-login-id [sending-terminal]
```

upon receipt of the initial message. Whenever a line of input as delimited by an NL, EOF, or EOL special character (see XBD Chapter 11, on page 199) is accumulated while in canonical input mode, the accumulated data shall be written on the other user's terminal. Characters shall be processed as follows:

- Typing <alert> shall write the <alert> character to the recipient's terminal.
- Typing the erase and kill characters shall affect the sender's terminal in the manner described by the **termios** interface in XBD Chapter 11 (on page 199).
- Typing the interrupt or end-of-file characters shall cause *write* to write an appropriate message (`"EOT\n"` in the POSIX locale) to the recipient's terminal and exit.
- Typing characters from *LC_CTYPE* classifications **print** or **space** shall cause those characters to be sent to the recipient's terminal.
- When and only when the *stty* **iexten** local mode is enabled, the existence and processing of additional special control characters and multi-byte or single-byte functions is implementation-defined.
- Typing other non-printable characters shall cause implementation-defined sequences of printable characters to be written to the recipient's terminal.

To write to a user who is logged in more than once, the *terminal* argument can be used to indicate which terminal to write to; otherwise, the recipient's terminal is selected in an implementation-defined manner and an informational message is written to the sender's standard output, indicating which terminal was chosen.

Permission to be a recipient of a *write* message can be denied or granted by use of the *mesg* utility. However, a user's privilege may further constrain the domain of accessibility of other users' terminals. The *write* utility shall fail when the user lacks appropriate privileges to perform the requested action.

OPTIONS

None.

OPERANDS

The following operands shall be supported:

user_name Login name of the person to whom the message shall be written. The application shall ensure that this operand is of the form returned by the *who* utility.

terminal Terminal identification in the same format provided by the *who* utility.

STDIN

Lines to be copied to the recipient's terminal are read from standard input.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *write*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files). If the recipient's locale does not use an *LC_CTYPE* equivalent to the sender's, the results are undefined.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of messages objects and message catalogs.

ASYNCHRONOUS EVENTS

If an interrupt signal is received, *write* shall write an appropriate message on the recipient's terminal and exit with a status of zero. It shall take the standard action for all other signals.

STDOUT

An informational message shall be written to standard output if a recipient is logged in more than once.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

The recipient's terminal is used for output.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 The addressed user is not logged on or the addressed user denies permission.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *talk* utility is considered by some users to be a more usable utility on full-screen terminals.

EXAMPLES

None.

RATIONALE

The *write* utility was included in this volume of POSIX.1-2024 since it can be implemented on all terminal types. The standard developers considered the *talk* utility, which cannot be implemented on certain terminals, to be a ``better'' communications interface. Both of these programs are in widespread use on historical implementations. Therefore, the standard developers decided that both utilities should be specified.

The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write* require that they all use or accept the same format.

FUTURE DIRECTIONS

None.

SEE ALSO

mesg, *talk*, *who*

XBD Chapter 8 (on page 167), Chapter 11 (on page 199)

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

The *write* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

Issue 8

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

NAME

xargs — construct argument lists and invoke utility

SYNOPSIS

XSI `xargs [-prtx] [-E eofstr|-0] [-I replstr|-L number|-n number] [-s size] [utility [argument...]]`

DESCRIPTION

The *xargs* utility shall construct a command line consisting of the *utility* and *argument* operands specified followed by as many arguments read in sequence from standard input as fit in length and number constraints specified by the options. The *xargs* utility shall then invoke the constructed command line and wait for its completion. This sequence shall be repeated until one of the following occurs:

- An end-of-file condition is detected on standard input.
- An argument consisting of just the logical end-of-file string (see the **−E** *eofstr* option) is found on standard input after double-quote processing, <apostrophe> processing, and <backslash>-escape processing (see next paragraph). All arguments up to but not including the argument consisting of just the logical end-of-file string shall be used as arguments in constructed command lines.
- An invocation of a constructed command line returns an exit status of 255.

If the **−0** option is not specified, the application shall ensure that arguments in the standard input are delimited by unquoted <blank> characters, unescaped <blank> characters, or <newline> characters, and quoting characters shall be interpreted as follows:

- A string of zero or more non-double-quote (`'"'`) non-<newline> characters can be quoted by enclosing them in double-quotes.
- A string of zero or more non-<apostrophe> (`'\''`) non-<newline> characters can be quoted by enclosing them in <apostrophe> characters.
- Any unquoted character can be escaped by preceding it with a <backslash>.

Multiple adjacent delimiter characters shall be treated as a single delimiter. If the standard input is not empty and does not end with a <newline>, the behavior is undefined (because the requirement in STDIN that the input is a text file is not met in that case).

If the **−0** option is specified, the application shall ensure that arguments in the standard input are delimited by null bytes. If multiple adjacent null bytes occur in the input, each null byte shall be treated as a delimiter. If the standard input is not empty and does not end with a null byte, *xargs* should ignore the trailing non-null bytes (as this can signal incomplete data) but may use them as the last argument passed to utility.

The utility named by *utility* shall be executed zero or more times until the end-of-file is reached or the logical end-of file string is found. If no arguments are supplied on standard input, the utility named by *utility* shall be executed zero times if the **−r** option is specified and shall be executed exactly once if the **−r** option is not specified. The results are unspecified if the utility named by *utility* attempts to read from its standard input.

The generated command line length shall be the sum of the size in bytes of the utility name and each argument treated as strings, including a null byte terminator for each of these strings. The *xargs* utility shall limit the command line length such that when the command line is invoked, the combined argument and environment lists (see the *exec* family of functions in the System Interfaces volume of POSIX.1-2024) shall not exceed {ARG_MAX}−2 048 bytes. Within this constraint, if neither the **−n** nor the **−s** option is specified, the default command line length shall be at least {LINE_MAX}.

OPTIONS

The *xargs* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–E *eofstr* Use *eofstr* as the logical end-of-file string. If neither **–E** nor **–0** is specified, it is unspecified whether the logical end-of-file string is the <underscore> character (`'_'`) or the end-of-file string capability is disabled. When *eofstr* is the null string, the logical end-of-file string capability shall be disabled and <underscore> characters shall be taken literally.

XSI **–I** *replstr* Insert mode: invoke *utility* for each argument from standard input. If **–0** is not specified, arguments in the standard input shall be delimited only by unescaped <newline> characters, not by <blank> characters, and any unquoted unescaped <blank> characters at the beginning of each line shall be ignored. The resulting argument shall be inserted in *arguments* in place of each occurrence of *replstr*. At least five arguments in *arguments* can each contain one or more instances of *replstr*. Each of these constructed arguments cannot grow larger than an implementation-defined limit greater than or equal to 255 bytes. Option **–x** shall be forced on.

XSI **–L** *number* Invoke *utility* for each set of *number* arguments from standard input. The last invocation of *utility* shall be with fewer arguments if fewer than *number* remain. If the **–0** option is not specified, each line in the standard input shall be treated as containing one argument except that empty lines shall be ignored and a line ending with a trailing unescaped <blank> shall signal continuation to the next non-empty line, inclusive; such continuation shall result in removal of all trailing unescaped <blank> characters and all <newline> characters that immediately follow them from the argument.

–n *number* Invoke *utility* using as many standard input arguments as possible, up to *number* (a positive decimal integer) arguments maximum. Fewer arguments shall be used if:

- The command line length accumulated exceeds the size specified by the **–s** option (or {LINE_MAX} if there is no **–s** option).
- The last iteration has fewer than *number*, but not zero, operands remaining.

–p Prompt mode: the user is asked whether to execute *utility* at each invocation. Trace mode (**–t**) is turned on to write the command instance to be executed, followed by a prompt to standard error. An affirmative response read from **/dev/tty** shall execute the command; otherwise, that particular invocation of *utility* shall be skipped.

–r Do not execute the utility named by *utility* if no arguments are supplied on standard input.

–s *size* Invoke *utility* using as many standard input arguments as possible yielding a command line length less than *size* (a positive decimal integer) bytes. Fewer arguments shall be used if:

- The total number of arguments exceeds that specified by the **–n** option.
- XSI The total number of arguments exceeds that specified by the **–L** option.
- End-of-file is encountered on standard input before *size* bytes are accumulated.

Values of *size* up to at least {LINE_MAX} bytes shall be supported, provided that the constraints specified in the DESCRIPTION are met. It shall not be considered

123236 an error if a value larger than that supported by the implementation or exceeding
123237 the constraints specified in the DESCRIPTION is given; *xargs* shall use the largest
123238 value it supports within the constraints.

123239 **−t** Enable trace mode. Each generated command line shall be written to standard
123240 error just prior to invocation.

123241 **−x** Terminate if a constructed command line will not fit in the implied or specified size
123242 (see the **−s** option above).

123243 **−0** Use a null byte as the input argument delimiter and do not treat any other input
123244 bytes as special.

123245 If the mutually exclusive **−0** and **−E** *eofstr* options are both specified, the behavior is unspecified,
123246 except that if *eofstr* is the null string the behavior shall be the same as if **−0** was specified without
123247 **−E** *eofstr*.

123248 **OPERANDS**

123249 The following operands shall be supported:

123250 *utility* The name of the utility to be invoked, found by search path using the *PATH*
123251 environment variable, described in XBD Chapter 8 (on page 167). If *utility* is
123252 omitted, the default shall be the *echo* utility. If the *utility* operand names any of the
123253 special built-in utilities in Section 2.15 (on page 2526), the results are undefined.

123254 *argument* An initial option or operand for the invocation of *utility*.

123255 **STDIN**

123256 If the **−0** option is not specified, the standard input shall be a text file and the results are
123257 unspecified if an end-of-file condition is detected immediately following an escaped <newline>.

123258 If the **−0** option is specified, the standard input need not be a text file, and *xargs* shall process the
123259 input as bytes, not characters.

123260 **INPUT FILES**

123261 The file **/dev/tty** shall be used to read responses required by the **−p** option.

123262 **ENVIRONMENT VARIABLES**

123263 The following environment variables shall affect the execution of *xargs*:

123264 *LANG* Provide a default value for the internationalization variables that are unset or null.
123265 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
123266 variables used to determine the values of locale categories.)

123267 *LC_ALL* If set to a non-empty string value, override the values of all the other
123268 internationalization variables.

123269 *LC_COLLATE*
123270 Determine the locale for the behavior of ranges, equivalence classes, and multi-
123271 character collating elements used in the extended regular expression defined for
123272 the **yesexpr** locale keyword in the *LC_MESSAGES* category.

123273 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
123274 characters (for example, single-byte as opposed to multi-byte characters in
123275 arguments and input files) and the behavior of character classes used in the
123276 extended regular expression defined for the **yesexpr** locale keyword in the
123277 *LC_MESSAGES* category.

123278 *LC_MESSAGES*
123279 Determine the locale used to process affirmative responses, and the locale used to
123280 affect the format and contents of diagnostic messages and prompts written to
123281 standard error.

123282 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

123283 *PATH* Determine the location of *utility*, as described in XBD Chapter 8 (on page 167).

123284 **ASYNCHRONOUS EVENTS**
123285 Default.

123286 **STDOUT**
123287 Not used.

123288 **STDERR**
123289 The standard error shall be used for diagnostic messages and the **–t** and **–p** options. If the **–t**
123290 option is specified, the *utility* and its constructed argument list shall be written to standard error,
123291 as it will be invoked, prior to invocation. If **–p** is specified, a prompt of the following format
123292 shall be written (in the POSIX locale):

123293 `"?..."`

123294 at the end of the line of the output from **–t**.

123295 **OUTPUT FILES**
123296 None.

123297 **EXTENDED DESCRIPTION**
123298 None.

123299 **EXIT STATUS**
123300 The following exit values shall be returned:

123301 0 Successful completion.

123302 1-125 A command line meeting the specified requirements could not be assembled, one or
123303 more of the invocations of *utility* returned a non-zero exit status, or some other error
123304 occurred.

123305 126 The utility specified by *utility* was found but could not be invoked.

123306 127 The utility specified by *utility* could not be found.

123307 **CONSEQUENCES OF ERRORS**
123308 If a command line meeting the specified requirements cannot be assembled, the utility cannot be
123309 invoked, an invocation of the utility is terminated by a signal, or an invocation of the utility exits
123310 with exit status 255, the *xargs* utility shall write a diagnostic message and exit without
123311 processing any remaining input.

123312 **APPLICATION USAGE**
123313 The 255 exit status allows a utility being used by *xargs* to tell *xargs* to terminate if it knows no
123314 further invocations using the current data stream will succeed. Thus, *utility* should explicitly *exit*
123315 with an appropriate value to avoid accidentally returning with 255.

123316 Note that since input is parsed as lines (if **–0** is not specified), with <blank> characters
123317 separating arguments and <backslash>, <apostrophe>, and double-quote characters used for
123318 quoting, if *xargs* is used to bundle the output of commands like *find dir* **–print** or *ls* into
123319 commands to be executed, unexpected results are likely if any filenames contain <blank>,
123320 <newline>, or quoting characters. This can be solved by using the **–print0** primary of *find*

123321 together with the *xargs* **−0** option, or by using *find* to call a script that converts each file found
123322 into a quoted string that is then piped to *xargs*, but in most cases it is preferable just to have *find*
123323 do the argument aggregation itself by using **−exec** with a `'+'` terminator instead of `';'`. Note
123324 that the quoting rules used by *xargs* are not the same as in the shell. They were not made
123325 consistent here because existing applications depend on the current rules. An easy (but
123326 inefficient) method that can be used to transform input consisting of one argument per line into
123327 a quoted form that *xargs* interprets correctly is to precede each non-<newline> character with a
123328 <backslash>. More efficient alternatives are shown in Example 2 and Example 5 below.

123329 On implementations with a large value for {ARG_MAX}, *xargs* may produce command lines
123330 longer than {LINE_MAX}. For invocation of utilities, this is not a problem. If *xargs* is being used
123331 to create a text file, users should explicitly set the maximum command line length with the **−s**
123332 option.

123333 The *command*, *env*, *nice*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit
123334 code 127 if a utility to be invoked cannot be found, so that applications can distinguish ``failure
123335 to find a utility'' from ``invoked utility exited with an error indication''. The value 127 was
123336 chosen because it is not commonly used for other meanings; most utilities use small values for
123337 ``normal error conditions'' and the values above 128 can be confused with termination due to
123338 receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could
123339 be found, but not invoked. Some scripts produce meaningful error messages differentiating the
123340 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice
123341 that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any
123342 attempt to *exec* the utility fails for any other reason.

123343 **EXAMPLES**

123344 1. The following command combines the output of the parenthesized commands (minus the
123345 <apostrophe> characters) onto one line, which is then appended to the file log. It assumes
123346 that the expansion of `"$0 $*"` does not include any <apostrophe> or <newline>
123347 characters.

123348 ```
(logname; date; printf "'%s'\n" "$0 $*") | xargs -E "" >>log
```

123349 2. The following command invokes *diff* with successive pairs of arguments originally typed
123350 as command line arguments.

123351 ```
printf "%s\0" "$@" | xargs -0 -n 2 -x diff --
```

123352 3. In the following command, the user is asked which regular files below the current
123353 directory are to be archived.

123354 ```
find . -type f -print0 | xargs -0 -p -L 1 ar -r arch
```

123355 4. The following command invokes *command1* one or more times with multiple arguments,
123356 stopping if an invocation of *command1* has a non-zero exit status.

123357 ```
xargs -E "" sh -c 'command1 "$@" || exit 255' sh < xargs_input
```

123358 **RATIONALE**

123359 The *xargs* utility was usually found only in System V-based systems; BSD systems included an
123360 *apply* utility that provided functionality similar to *xargs* **−n** *number*. The SVID lists *xargs* as a
123361 software development extension. This volume of POSIX.1-2024 does not share the view that it is
123362 used only for development, and therefore it is not optional.

123363 The classic application of the *xargs* utility is in conjunction with the *find* utility to reduce the
123364 number of processes launched by a simplistic use of the *find* **−exec** combination. The *xargs* utility
123365 is also used to enforce an upper limit on memory required to launch a process. With this basis in

mind, this volume of POSIX.1-2024 selected only the minimal features required.

Although the 255 exit status is mostly an accident of historical implementations, it allows a utility being used by *xargs* to tell *xargs* to terminate if it knows no further invocations using the current data stream shall succeed. Any non-zero exit status from a utility falls into the 1-125 range when *xargs* exits. There is no statement of how the various non-zero utility exit status codes are accumulated by *xargs*. The value could be the addition of all codes, their highest value, the last one received, or a single value such as 1. Since no algorithm is arguably better than the others, and since many of the standard utilities say little more (portably) than ``pass/fail'', no new algorithm was invented.

Several other *xargs* options were removed because simple alternatives already exist within this volume of POSIX.1-2024. For example, the **−i** *replstr* option can be just as efficiently performed using a shell **for** loop. Since *xargs* calls an *exec* function with each input line, the **−i** option does not usually exploit the grouping capabilities of *xargs*.

The requirement that *xargs* never produces command lines such that invocation of *utility* is within 2 048 bytes of hitting the POSIX *exec* {ARG_MAX} limitations is intended to guarantee that the invoked utility has room to modify its environment variables and command line arguments and still be able to invoke another utility. Note that the minimum {ARG_MAX} allowed by the System Interfaces volume of POSIX.1-2024 is 4 096 bytes and the minimum value allowed by this volume of POSIX.1-2024 is 2 048 bytes; therefore, the 2 048 bytes difference seems reasonable. Note, however, that *xargs* may never be able to invoke a utility if the environment passed in to *xargs* comes close to using {ARG_MAX} bytes.

The version of *xargs* required by this volume of POSIX.1-2024 is required to wait for the completion of the invoked command before invoking another command. This was done because historical scripts using *xargs* assumed sequential execution. Implementations wanting to provide parallel operation of the invoked utilities are encouraged to add an option enabling parallel invocation, but should still wait for termination of all of the children before *xargs* terminates normally.

The **−e** option was omitted from the ISO POSIX-2: 1993 standard in the belief that the *eofstr* option-argument was recognized only when it was on a line by itself and before quote and escape processing were performed, and that the logical end-of-file processing was only enabled if a **−e** option was specified. In that case, a simple *sed* script could be used to duplicate the **−e** functionality. Further investigation revealed that:

- The logical end-of-file string was checked for after quote and escape processing, making a *sed* script that provided equivalent functionality much more difficult to write.
- The default was to perform logical end-of-file processing with an <underscore> as the logical end-of-file string.

To correct this misunderstanding, the **−E** *eofstr* option was adopted from the X/Open Portability Guide. Users should note that the description of the **−E** option matches historical documentation of the **−e** option (which was not adopted because it did not support the Utility Syntax Guidelines), by saying that if *eofstr* is the null string, logical end-of-file processing is disabled. Historical implementations of *xargs* actually did not disable logical end-of-file processing; they treated a null argument found in the input as a logical end-of-file string. (A null *string* argument could be generated using single or double-quotes (`''` or `""`). Since this behavior was not documented historically, it is considered to be a bug.

The **−I**, **−L**, and **−n** options are mutually-exclusive. Some implementations use the last one specified if more than one is given on a command line; other implementations treat combinations of the options in different ways.

FUTURE DIRECTIONS

A future version of this standard may require that, when the **–0** option is specified, if the standard input is not empty and does not end with a null byte, *xargs* ignores the trailing non-null bytes.

SEE ALSO

Chapter 2 (on page 2472), *diff*, *echo*, *find*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *exec*

CHANGE HISTORY

First released in Issue 2.

Issue 5

A second FUTURE DIRECTION is added.

Issue 6

The obsolescent **–e**, **–i**, and **–l** options are removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **–p** option is added.
- In the INPUT FILES section, the file **/dev/tty** is used to read responses required by the **–p** option.
- The STDERR section is updated to describe the **–p** option.

The description of the **–E** option is aligned with the ISO POSIX-2: 1993 standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #123 is applied, changing the description of the *xargs* **–I** option.

Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the *LC_MESSAGES* environment variable.

SD5-XCU-ERN-68 is applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

SD5-XCU-ERN-128 is applied, clarifying the DESCRIPTION of the logical end-of-file string.

SD5-XCU-ERN-132 is applied, updating the EXAMPLES section.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0149 [342] is applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0203 [499] is applied.

Issue 8

Austin Group Defect 243 is applied, adding the **–r** and **–0** options.

Austin Group Defect 248 is applied, changing the EXAMPLES section.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1586 is applied, adding the *timeout* utility.

123451 Austin Group Defect 1594 is applied, changing the APPLICATION USAGE section.

NAME

xgettext — extract *gettext* call strings from C-language source files (DEVELOPMENT)

SYNOPSIS

CD

```
xgettext [-j] [-n] [-d default-domain] [-K keyword-spec]...
    [-p pathname] file...

xgettext -a [-n] [-d default-domain] [-p pathname]
    [-x exclude-file] file...
```

DESCRIPTION

The *xgettext* utility shall automate the creation of portable messages object source files (dot-po files). A dot-po file shall contain copies of string literals that are found in C-language source code in files specified by *file* operands. The dot-po file can be used as input to the *msgfmt* utility, to produce a messages object file that can be used by applications.

The *xgettext* utility shall write *msgid* argument strings that are passed as string literals in *gettext*(), *gettext_l*(), *ngettext*(), and *ngettext_l*() calls in C-language source code to the default output file; this file shall be named **messages.po** unless it is changed by the **–d** option. The *xgettext* utility shall also write *msgid* argument strings that are passed as string literals in *dcgettext*(), *dcgettext_l*(), *dcngettext*(), *dcngettext_l*(), *dgettext*(), *dgettext_l*(), *dngettext*(), and *dngettext_l*() calls either to the default output file or to the output file *domainname***.po** where *domainname* is the first parameter to the call; it is implementation-defined which of those output files is used. A **msgid** directive shall precede each *msgid* argument string. For the functions that have a *msgid_plural* argument, a **msgid_plural** directive followed by that argument string shall also be written directly after the corresponding **msgid** directive. A **msgstr** directive or **msgstr[***index***]** directives with an empty string shall be written after the corresponding **msgid** or **msgid_plural** directive, respectively. The function names that *xgettext* searches for can be changed using the **–K** option.

The first directive in each created dot-po file shall be a **domain** directive giving the associated domain name, except that this directive is optional in the default output file.

If the **–p** *pathname* option is specified, *xgettext* shall create the dot-po files in the *pathname* directory. Otherwise, the dot-po files shall be created in the current working directory.

The **msgid** values shall be in the same order that the strings are extracted from each *file* and subsections with duplicate **msgid** values shall be written to the dot-po files as comment lines.

OPTIONS

The *xgettext* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

–a Extract all strings, not just those found in calls to *gettext* family functions. Only one dot-po file shall be created.

–d *default-domain*
Name the default output file *default-domain***.po** instead of **messages.po**.

–j Join messages from C-language source files with existing dot-po files. For each dot-po file that *xgettext* writes messages to, if the file does not exist, it shall be created. New messages shall be appended but any subsections with duplicate **msgid** values except the first (including **msgid** values found in an existing dot-po file) shall either be commented out or omitted in the resulting dot-po file; if omitted, a warning message may be written to standard error. Domain directives in the existing dot-po files shall be ignored; the assumption is that all previous

msgid values belong to the same domain. The behavior is unspecified if an existing dot-po file was not created by *xgettext* or has been modified by another application.

−K *keyword-spec*
Specify an additional keyword to be looked for:

- If *keyword-spec* is an empty string, this shall disable the use of default keywords for the *gettext* family of functions.
- If *keyword-spec* is a C identifier, *xgettext* shall look for strings in the first argument of each call to the function or macro *keyword-spec*.
- If *keyword-spec* is of the form *id*:*argnum* then *xgettext* shall treat the *argnum*-th argument of a call to the function or macro *id* as the *msgid* argument, where *argnum* 1 is the first argument.
- If *keyword-spec* is of the form *id*:*argnum1*,*argnum2* then *xgettext* shall treat strings in the *argnum1*-th argument and in the *argnum2*-th argument of a call to the function or macro *id* as the *msgid* and *msgid_plural* arguments, respectively.

For all mentioned forms, the application shall ensure that if *argnum2* is given, it is not equal to *argnum1*. All numeric values shall be converted as specified in item 6 in XBD Section 12.1 (on page 213).

−n Add comment lines to the output file indicating pathnames and line numbers in the source files where each extracted string is encountered. These lines shall appear before each **msgid** directive. Such comments should have the format:

```
#: pathname1:linenumber1 [pathname2:linenumber2...]
```

−p *pathname*
Create output files in the directory specified by *pathname* instead of in the current working directory.

−x *exclude-file*
Specify a file containing strings that shall not be extracted from the input files. The format of *exclude-file* is identical to that of a dot-po file. However, only statements containing **msgid** directives in *exclude-file* shall be used. All other statements shall be ignored.

OPERANDS

The following operand shall be supported:

file A pathname of an input file containing C-language source code. If `'-'` is specified for an instance of *file*, the standard input shall be used.

STDIN

The standard input shall not be used unless a *file* operand is specified as `'-'`.

INPUT FILES

The input files specified as *file* operands shall be C-language source files. The input file specified as the option-argument for the **−x** option shall be a dot-po file in the format specified as input for the *msgfmt* utility.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *xgettext*:

123539 *LANG* Provide a default value for the internationalization variables that are unset or null.
123540 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
123541 variables used to determine the values of locale categories.)

123542 XSI *LANGUAGE* Determine the location of messages objects if *NLSPATH* is not set or the evaluation
123543 of *NLSPATH* did not lead to a suitable messages object being found.

123544 *LC_ALL* If set to a non-empty string value, override the values of all the other
123545 internationalization variables.

123546 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
123547 characters (for example, single-byte as opposed to multi-byte characters in
123548 arguments and input files).

123549 *LC_MESSAGES*
123550 Determine the locale name used to locate messages objects, and the locale that
123551 should be used to affect the format and contents of diagnostic messages written to
123552 standard error.

123553 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

123554 **ASYNCHRONOUS EVENTS**
123555 Default.

123556 **STDOUT**
123557 The standard output shall not be used.

123558 **STDERR**
123559 The standard error shall be used for diagnostic messages and may be used for warning
123560 messages.

123561 **OUTPUT FILES**
123562 The output files shall be dot-po files in the format specified as input for the *msgfmt* utility. It is
123563 unspecified whether each output file includes a header (**msgid ""**) before the content derived
123564 from the input C-language source files.

123565 **EXTENDED DESCRIPTION**
123566 None.

123567 **EXIT STATUS**
123568 The following exit values shall be returned:

123569 0 Successful completion.

123570 >0 An error occurred.

123571 **CONSEQUENCES OF ERRORS**
123572 Default.

123573 **APPLICATION USAGE**
123574 Implementations differ as to whether they write all output to the default output file or split the
123575 output into separate per-domain files. Portable applications can either ensure that each C-
123576 language source file contains calls to *gettext* family functions for only a single domain, or force
123577 all output to be to the default output file by using the **−K** option to override the default
123578 keywords.

123579 Some implementations of *xgettext* are not able to extract cast strings (unless **−a** is used), for
123580 example casts of literal strings to **(const char *)**. Use of a cast is unnecessary anyway, since the
123581 prototypes in **<libintl.h>** already specify this type.

The *xgettext* utility is not required to handle C preprocessor directives. Therefore if, for example, calls to *gettext* family functions are wrapped by macros, they might not be found unless the **–K** option is used to tell *xgettext* to look for the macro calls.

EXAMPLES

Example 1

The following example shows how **–K** can be used to force all output to be to the default output file:

```
xgettext –K "" –K gettext:1 –K dgettext:2 –K dcgettext:2 \
    –K ngettext:1,2 –K dngettext:2,3 –K dcngettext:2,3 source.c
```

By overriding the default keywords using the **–K** option as above, the *xgettext* utility is directed to ignore the *domainname* arguments to the *dgettext*(), *dcgettext*(), *dngettext*(), and *dcngettext*() functions. Thus, the utility treats the functions as their respective equivalent without the *d* prefix, ignoring the *domainname* argument and writing generated output to the default output file, **messages.po**. Additional **–K** options would be needed for the variants of the functions with an *_l* suffix if they are used.

Example 2

If the source uses a macro definition such as:

```
#define i18n gettext
```

the use of:

```
xgettext –K i18n:1 source.c
```

will pick up **msgid** values from a line such as:

```
fprintf(stdout, i18n("The value is %s"), value1);
```

RATIONALE

The **–K** option is based on the **–k** option of GNU *xgettext*; the only difference is that GNU's **–k** takes an optional option-argument whereas **–K** in this standard has a mandatory option-argument in order to comply with the syntax guidelines.

The standard developers considered including functionality equivalent to the **–c**, **–m**, and **–M** options in existing implementations. However, those letters could not be used as the syntax differed between implementations. The usual solution of adding an uppercase equivalent of lowercase options with the standard syntax instead was not possible, for obvious reasons for **–m** and **–M**, and as **–C** was already in use for another purpose in one implementation.

The **–s** option is not included as it has been deprecated in at least one implementation because it has been found to deprive translators of valuable context.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

A future version of this standard may change the description of the **–n** option to mandate the given comment format (by using ``shall'' instead of ``should'').

SEE ALSO

gettext, *msgfmt*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

XSH *gettext*

CHANGE HISTORY

First released in Issue 8.

NAME

yacc — yet another compiler compiler (**DEVELOPMENT**)

SYNOPSIS

CD

```
yacc [-dltv] [-b file_prefix] [-p sym_prefix] grammar
```

DESCRIPTION

The *yacc* utility shall read a description of a context-free grammar in *grammar* and write C source code, conforming to the ISO C standard, to a code file, and optionally header information into a header file, in the current directory. The generated source code shall not depend on any undefined, unspecified, or implementation-defined behavior, except in cases where it is copied directly from the supplied grammar, or in cases that are documented by the implementation. The C code shall define a function and related routines and macros for an automaton that executes a parsing algorithm meeting the requirements in Algorithms (on page 3625).

The form and meaning of the grammar are described in the EXTENDED DESCRIPTION section.

The C source code and header file shall be produced in a form suitable as input for the C compiler (see *c17*).

OPTIONS

The *yacc* utility shall conform to XBD Section 12.2 (on page 215), except for Guideline 9.

The following options shall be supported:

−b *file_prefix* Use *file_prefix* instead of **y** as the prefix for all output filenames. The code file **y.tab.c**, the header file **y.tab.h** (created when **−d** is specified), and the description file **y.output** (created when **−v** is specified), shall be changed to *file_prefix***.tab.c**, *file_prefix***.tab.h**, and *file_prefix***.output**, respectively.

−d Write the header file; by default only the code file is written. See the OUTPUT FILES section.

−l Produce a code file that does not contain any **#line** constructs. If this option is not present, it is unspecified whether the code file or header file contains **#line** directives. This should only be used after the grammar and the associated actions are fully debugged.

−p *sym_prefix*
Use *sym_prefix* instead of **yy** as the prefix for all external names produced by *yacc*. The names affected shall include the functions *yyparse*( ), *yylex*( ), and *yyerror*( ), and the variables *yylval*, *yychar*, and *yydebug*. (In the remainder of this section, the six symbols cited are referenced using their default names only as a notational convenience.) Local names may also be affected by the **−p** option; however, the **−p** option shall not affect **#define** symbols generated by *yacc*.

−t Modify conditional compilation directives to permit compilation of debugging code in the code file. Runtime debugging statements shall always be contained in the code file, but by default conditional compilation directives prevent their compilation.

−v Write a file containing a description of the parser and a report of conflicts generated by ambiguities in the grammar.

123668 **OPERANDS**

123669 The following operand is required:

123670 *grammar* A pathname of a file containing instructions, hereafter called *grammar*, for which a
123671 parser is to be created. The format for the grammar is described in the EXTENDED
123672 DESCRIPTION section.

123673 **STDIN**

123674 Not used.

123675 **INPUT FILES**

123676 The file *grammar* shall be a text file formatted as specified in the EXTENDED DESCRIPTION
123677 section.

123678 **ENVIRONMENT VARIABLES**

123679 The following environment variables shall affect the execution of *yacc*:

123680 *LANG* Provide a default value for the internationalization variables that are unset or null.
123681 (See XBD Section 8.2 (on page 169) for the precedence of internationalization
123682 variables used to determine the values of locale categories.)

123683 *LC_ALL* If set to a non-empty string value, override the values of all the other
123684 internationalization variables.

123685 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
123686 characters (for example, single-byte as opposed to multi-byte characters in
123687 arguments and input files).

123688 *LC_MESSAGES*

123689 Determine the locale that should be used to affect the format and contents of
123690 diagnostic messages written to standard error.

123691 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

123692 The *LANG* and *LC_** variables affect the execution of the *yacc* utility as stated. The *main*()
123693 function defined in Yacc Library (on page 3624) shall call:

123694 `setlocale(LC_ALL, "")`

123695 and thus the program generated by *yacc* shall also be affected by the contents of these variables
123696 at runtime.

123697 **ASYNCHRONOUS EVENTS**

123698 Default.

123699 **STDOUT**

123700 Not used.

123701 **STDERR**

123702 If shift/reduce or reduce/reduce conflicts are detected in *grammar*, *yacc* shall write a report of
123703 those conflicts to the standard error in an unspecified format.

123704 Standard error shall also be used for diagnostic messages.

123705 **OUTPUT FILES**

123706 The code file, the header file, and the description file shall be text files. All are described in the
123707 following sections.

123708 **Code File**

123709 This file shall contain the C source code for the *yyparse*( ) function. It shall contain code for the
123710 various semantic actions with macro substitution performed on them as described in the
123711 EXTENDED DESCRIPTION section. Preceding this code it shall contain an `extern int
123712 yychar` declaration or `int yychar` definition, and **#define** statements for the following
123713 macros:

123714 YYEMPTY Token number indicating there is no lookahead token. This macro shall expand to
123715 an integer constant with a value less than zero, protected by parentheses.

123716 YYEOF Token number indicating the end of input. This macro shall expand to the value 0.

123717 It also shall contain a copy of the **#define** statements in the header file, prior to any code copied
123718 from semantic actions in *grammar*, and the following function prototypes for the *yyerror*( ),
123719 *yylex*( ), and *yyparse*( ) functions, after any code copied from within **%{** and **%}** in the declarations
123720 section in *grammar* and before any code copied from semantic actions in *grammar*:

```
123721 void yyerror(const char *);
123722 int  yylex(void);
123723 int  yyparse(void);
```

123724 The declarations of *yyerror*( ) and *yylex*( ) shall be protected by **#ifndef** or **#if** preprocessor
123725 statements such that each is only visible if a preprocessor macro with the name yyerror or yylex,
123726 respectively, is not already defined, where the yy in the macro names is replaced by *sym_prefix* if
123727 the **−p** *sym_prefix* option is used.

123728 If a **%union** declaration is used, the declaration for YYSTYPE and an `extern YYSTYPE
123729 yylval` declaration or `YYSTYPE yylval` definition shall also be included in this file.

123730 The code file shall not contain a declaration of the *main*( ) function, unless one is present within
123731 **%{** and **%}** in the declarations section in *grammar*.

123732 **Header File**

123733 The header file shall contain **#define** statements that associate the token numbers with the token
123734 names. This allows source files other than the code file to access the token codes. If a **%union**
123735 declaration is used, the declaration for YYSTYPE and an `extern YYSTYPE yylval`
123736 declaration shall also be included in this file. The header file may also declare the *yyparse*( )
123737 function, using a function prototype. It shall not declare the *yyerror*( ) and *yylex*( ) functions.

123738 **Description File**

123739 The description file shall be a text file containing a description of the state machine
123740 corresponding to the parser, using an unspecified format. Limits for internal tables (see Limits,
123741 on page 3625) shall also be reported, in an implementation-defined manner. (Some
123742 implementations may use dynamic allocation techniques and have no specific limit values to
123743 report.)

123744 **EXTENDED DESCRIPTION**

123745 The *yacc* command accepts a language that is used to define a grammar for a target language to
123746 be parsed by the tables and code generated by *yacc*. The language accepted by *yacc* as a
123747 grammar for the target language is described below using the *yacc* input language itself.

123748 The input *grammar* includes rules describing the input structure of the target language and code
123749 to be invoked when these rules are recognized to provide the associated semantic action. The
123750 code to be executed shall appear as bodies of text that are intended to be C-language code. These
123751 bodies of text shall not contain C-language trigraphs. The C-language inclusions are presumed

123752 to form a correct function when processed by *yacc* into its output files. The code included in this
123753 way shall be executed during the recognition of the target language.

123754 Given a grammar, the *yacc* utility generates the files described in the OUTPUT FILES section.
123755 The code file can be compiled and linked using *c17*. If the declaration and programs sections of
123756 the grammar file did not include definitions of *main*(), *yylex*(), and *yyerror*(), the compiled
123757 output requires linking with externally supplied versions of those functions. Default versions of
123758 *main*() and *yyerror*() are supplied in the *yacc* library and can be linked in by using the **–l y**
123759 operand to *c17*. The *yacc* library interfaces need not support interfaces with other than the
123760 default **yy** symbol prefix. The application provides the lexical analyzer function, *yylex*(); the *lex*
123761 utility is specifically designed to generate such a routine.

123762 **Input Language**

123763 The application shall ensure that every specification file consists of three sections in order:
123764 *declarations*, *grammar rules*, and *programs*, separated by double <percent-sign> characters (`"%%"`).
123765 The declarations and programs sections can be empty. If the latter is empty, the preceding `"%%"`
123766 mark separating it from the rules section can be omitted.

123767 The input is free form text following the structure of the grammar defined below.

123768 **Lexical Structure of the Grammar**

123769 The <blank>, <newline>, and <form-feed> character shall be ignored, except that the application
123770 shall ensure that they do not appear in names or multi-character reserved symbols. Comments
123771 shall be enclosed in `"/* ... */"`, and can appear wherever a name is valid.

123772 Names are of arbitrary length, made up of letters, periods (`'.'`), underscores (`'_'`), and non-
123773 initial digits. Uppercase and lowercase letters are distinct. Conforming applications shall not
123774 use names beginning in **yy** or **YY** since the *yacc* parser uses such names. Many of the names
123775 appear in the final output of *yacc*, and thus they should be chosen to conform with any
123776 additional rules created by the C compiler to be used. In particular they appear in **#define**
123777 statements.

123778 A literal shall consist of a single character enclosed in single-quote characters. All of the escape
123779 sequences supported for character constants by the ISO C standard shall be supported by *yacc*.

123780 The relationship with the lexical analyzer is discussed in detail below.

123781 The application shall ensure that the NUL character is not used in grammar rules or literals.

123782 **Declarations Section**

123783 The declarations section is used to define the symbols used to define the target language and
123784 their relationship with each other. In particular, much of the additional information required to
123785 resolve ambiguities in the context-free grammar for the target language is provided here.

123786 Usually *yacc* assigns the relationship between the symbolic names it generates and their
123787 underlying numeric value. The declarations section makes it possible to control the assignment
123788 of these values.

123789 It is also possible to keep semantic information associated with the tokens currently on the parse
123790 stack in a user-defined C-language **union**, if the members of the union are associated with the
123791 various names in the grammar. The declarations section provides for this as well.

123792 The first group of declarators below all take a list of names as arguments. That list can optionally
123793 be preceded by the name of a C union member (called a *tag* below) appearing within `'<'` and
123794 `'>'`. (As an exception to the typographical conventions of the rest of this volume of

123795 POSIX.1-2024, in this case **<***tag***>** does not represent a metavariable, but the literal angle bracket
123796 characters surrounding a symbol.) The use of *tag* specifies that the tokens named on this line
123797 shall be of the same C type as the union member referenced by *tag*. This is discussed in more
123798 detail below.

123799 For lists used to define tokens, the first appearance of a given token can be followed by a
123800 positive integer (as a string of decimal digits). If this is done, the underlying value assigned to it
123801 for lexical purposes shall be taken to be that number.

123802 The following declares *name* to be a token:

123803 `%token [<tag>] name [number] [name [number]]...`

123804 If *tag* is present, the C type for all tokens on this line shall be declared to be the type referenced
123805 by *tag*. If a positive integer, *number*, follows a *name*, that value shall be assigned to the token.

123806 The following declares *name* to be a token, and assigns precedence to it:

123807 `%left [<tag>] name [number] [name [number]]...`
123808 `%right [<tag>] name [number] [name [number]]...`

123809 One or more lines, each beginning with one of these symbols, can appear in this section. All
123810 tokens on the same line have the same precedence level and associativity; the lines are in order
123811 of increasing precedence or binding strength. **%left** denotes that the operators on that line are
123812 left associative, and **%right** similarly denotes right associative operators. If *tag* is present, it shall
123813 declare a C type for *name*s as described for **%token**.

123814 The following declares *name* to be a token, and indicates that this cannot be used associatively:

123815 `%nonassoc [<tag>] name [number] [name [number]]...`

123816 If the parser encounters associative use of this token it reports an error. If *tag* is present, it shall
123817 declare a C type for *name*s as described for **%token**.

123818 The following declares that union member *name*s are non-terminals, and thus it is required to
123819 have a *tag* field at its beginning:

123820 `%type <tag> name...`

123821 Because it deals with non-terminals only, assigning a token number or using a literal is also
123822 prohibited. If this construct is present, *yacc* shall perform type checking; if this construct is not
123823 present, the parse stack shall hold only the **int** type.

123824 Every name used in *grammar* not defined by a **%token**, **%left**, **%right**, or **%nonassoc** declaration
123825 is assumed to represent a non-terminal symbol. The *yacc* utility shall report an error for any non-
123826 terminal symbol that does not appear on the left side of at least one grammar rule.

123827 Once the type, precedence, or token number of a name is specified, it shall not be changed. If the
123828 first declaration of a token does not assign a token number, *yacc* shall assign a token number.
123829 Once this assignment is made, the token number shall not be changed by explicit assignment.

123830 The following declarators do not follow the previous pattern.

123831 The following declares the non-terminal *name* to be the *start symbol*, which represents the largest,
123832 most general structure described by the grammar rules:

123833 `%start name`

123834 By default, it is the left-hand side of the first grammar rule; this default can be overridden with
123835 this declaration.

123836 The following declares the *yacc* value stack to be a union of the various types of values desired.

123837 `%union { body of union (in C) }`

123838 The body of the union shall not contain unbalanced curly brace preprocessing tokens.

123839 By default, the values returned by actions (see below) and the lexical analyzer shall be of type
123840 **int**. The *yacc* utility keeps track of types, and it shall insert corresponding union member names
123841 in order to perform strict type checking of the resulting parser.

123842 Alternatively, given that at least one <*tag*> construct is used, the union can be declared in a
123843 header file (which shall be included in the declarations section by using a **#include** construct
123844 within **%{** and **%}**), and a **typedef** used to define the symbol YYSTYPE to represent this union.
123845 The effect of **%union** is to provide the declaration of YYSTYPE directly from the *yacc* input.

123846 C-language declarations and definitions can appear in the declarations section, enclosed by the
123847 following marks:

123848 `%{ ... %}`

123849 These statements shall be copied into the code file, and have global scope within it so that they
123850 can be used in the rules and program sections. The statements shall not contain `"%}"` outside a
123851 comment, string literal, or multi-character constant.

123852 The application shall ensure that the declarations section is terminated by the token **%%**.

123853 **Grammar Rules in yacc**

123854 The rules section defines the context-free grammar to be accepted by the function *yacc* generates,
123855 and associates with those rules C-language actions and additional precedence information. The
123856 grammar is described below, and a formal definition follows.

123857 The rules section is comprised of one or more grammar rules. A grammar rule has the form:

123858 `A : BODY ;`

123859 The symbol **A** represents a non-terminal name, and **BODY** represents a sequence of zero or
123860 more *name*s, *literal*s, and *semantic action*s that can then be followed by optional *precedence rule*s.
123861 Only the names and literals participate in the formation of the grammar; the semantic actions
123862 and precedence rules are used in other ways. The <colon> and the <semicolon> are *yacc*
123863 punctuation. If there are several successive grammar rules with the same left-hand side, the
123864 <vertical-line> (`'|'`) can be used to avoid rewriting the left-hand side; in this case the
123865 <semicolon> appears only after the last rule. The BODY part can be empty (or empty of names
123866 and literals) to indicate that the non-terminal symbol matches the empty string.

123867 The *yacc* utility assigns a unique number to each rule. Rules using the vertical bar notation are
123868 distinct rules. The number assigned to the rule appears in the description file.

123869 The elements comprising a BODY are:

123870 *name*, *literal* These form the rules of the grammar: *name* is either a *token* or a *non-terminal*; *literal*
123871 stands for itself (less the lexically required quotation marks).

123872 *semantic action*

123873 With each grammar rule, the user can associate actions to be performed each time
123874 the rule is recognized in the input process. (Note that the word ``action'' can also
123875 refer to the actions of the parser—shift, reduce, and so on.)

123876 These actions can return values and can obtain the values returned by previous
123877 actions. These values are kept in objects of type YYSTYPE (see **%union**). The
123878 result value of the action shall be kept on the parse stack with the left-hand side of
123879 the rule, to be accessed by other reductions as part of their right-hand side. By

123880 using the <*tag*> information provided in the declarations section, the code
123881 generated by *yacc* can be strictly type checked and contain arbitrary information. In
123882 addition, the lexical analyzer can provide the same kinds of values for tokens, if
123883 desired.

123884 An action is an arbitrary C statement and as such can do input or output, call
123885 subprograms, and alter external variables. An action is one or more C statements
123886 enclosed in curly braces `'{'` and `'}'`. The statements shall not contain
123887 unbalanced curly brace preprocessing tokens.

123888 Certain pseudo-variables can be used in the action. These are macros for access to
123889 data structures known internally to *yacc*.

123890 $$ The value of the action can be set by assigning it to $$. If type
123891 checking is enabled and the type of the value to be assigned cannot
123892 be determined, a diagnostic message may be generated.

123893 $*number* This refers to the value returned by the component specified by the
123894 token *number* in the right side of a rule, reading from left to right;
123895 *number* can be zero or negative. If *number* is zero or negative, it refers
123896 to the data associated with the name on the parser's stack preceding
123897 the leftmost symbol of the current rule. (That is, `"$0"` refers to the
123898 name immediately preceding the leftmost name in the current rule to
123899 be found on the parser's stack and `"$-1"` refers to the symbol to *its*
123900 left.) If *number* refers to an element past the current point in the rule,
123901 or beyond the bottom of the stack, the result is undefined. If type
123902 checking is enabled and the type of the value to be assigned cannot
123903 be determined, a diagnostic message may be generated.

123904 $<*tag*>*number*
123905 These correspond exactly to the corresponding symbols without the
123906 *tag* inclusion, but allow for strict type checking (and preclude
123907 unwanted type conversions). The effect is that the macro is expanded
123908 to use *tag* to select an element from the YYSTYPE union (using
123909 *dataname.tag*). This is particularly useful if *number* is not positive.

123910 $<*tag*>$ This imposes on the reference the type of the union member
123911 referenced by *tag*. This construction is applicable when a reference to
123912 a left context value occurs in the grammar, and provides *yacc* with a
123913 means for selecting a type.

123914 Actions can occur anywhere in a rule (not just at the end); an action can access
123915 values returned by actions to its left, and in turn the value it returns can be
123916 accessed by actions to its right. An action appearing in the middle of a rule shall be
123917 equivalent to replacing the action with a new non-terminal symbol and adding an
123918 empty rule with that non-terminal symbol on the left-hand side. The semantic
123919 action associated with the new rule shall be equivalent to the original action. The
123920 use of actions within rules might introduce conflicts that would not otherwise
123921 exist.

123922 By default, the value of a rule shall be the value of the first element in it. If the first
123923 element does not have a type (particularly in the case of a literal) and type
123924 checking is turned on by **%type**, an error message shall result.

123925 *precedence* The keyword **%prec** can be used to change the precedence level associated with a
123926 particular grammar rule. Examples of this are in cases where a unary and binary
123927 operator have the same symbolic representation, but need to be given different
123928 precedences, or where the handling of an ambiguous if-else construction is
123929 necessary. The reserved symbol **%prec** can appear immediately after the body of
123930 the grammar rule and can be followed by a token name or a literal. It shall cause
123931 the precedence of the grammar rule to become that of the following token name or
123932 literal. The action for the rule as a whole can follow **%prec**.

123933 If a program section follows, the application shall ensure that the grammar rules are terminated
123934 by **%%**.

123935 **Programs Section**

123936 The *programs* section can include the definition of the lexical analyzer *yylex*(), and any other
123937 functions; for example, those used in the actions specified in the grammar rules. It is unspecified
123938 whether the programs section precedes or follows the semantic actions in the output file;
123939 therefore, if the application contains any macro definitions and declarations intended to apply to
123940 the code in the semantic actions, it shall place them within `"%{ ... %}"` in the declarations
123941 section.

123942 **Input Grammar**

123943 The following input to *yacc* yields a parser for the input to *yacc*. This formal syntax takes
123944 precedence over the preceding text syntax description.

123945 The lexical structure is defined less precisely; Lexical Structure of the Grammar (on page 3616)
123946 defines most terms. The correspondence between the previous terms and the tokens below is as
123947 follows.

123948 **IDENTIFIER** This corresponds to the concept of *name*, given previously. It also includes
123949 literals as defined previously.

123950 **C_IDENTIFIER** This is a name, and additionally it is known to be followed by a <colon>. A
123951 literal cannot yield this token.

123952 **NUMBER** A string of digits (a non-negative decimal integer).

123953 **TYPE**, **LEFT**, **MARK**, **LCURL**, **RCURL**
123954 These correspond directly to **%type**, **%left**, **%%**, **%{**, and **%}**.

123955 **{...}** This indicates C-language source code, with the possible inclusion of `'$'`
123956 macros as discussed previously.

```
123957 /* Grammar for the input to yacc. */
123958 /* Basic entries. */
123959 /* The following are recognized by the lexical analyzer. */

123960 %token    IDENTIFIER      /* Includes identifiers and literals */
123961 %token    C_IDENTIFIER    /* identifier (but not literal)
123962                              followed by a :. */
123963 %token    NUMBER          /* [0-9][0-9]* */

123964 /* Reserved words : %type=>TYPE %left=>LEFT, and so on */

123965 %token    LEFT RIGHT NONASSOC TOKEN PREC TYPE START UNION

123966 %token    MARK            /* The %% mark. */
123967 %token    LCURL           /* The %{ mark. */
```



```
%token    RCURL           /* The %} mark. */

/* 8-bit character literals stand for themselves; */
/* tokens have to be defined for multi-byte characters. */

%start    spec

%%

spec  : defs MARK rules tail
      ;
tail  : MARK
      {
        /* In this action, set up the rest of the file. */
      }
      | /* Empty; the second MARK is optional. */
      ;
defs  : /* Empty. */
      |    defs def
      ;
def   : START IDENTIFIER
      |    UNION
      {
        /* Copy union definition to output. */
      }
      |    LCURL
      {
        /* Copy C code to output file. */
      }
        RCURL
      |    rword tag nlist
      ;
rword : TOKEN
      | LEFT
      | RIGHT
      | NONASSOC
      | TYPE
      ;
tag   : /* Empty: union tag ID optional. */
      | '<' IDENTIFIER '>'
      ;
nlist : nmno
      | nlist nmno
      ;
nmno  : IDENTIFIER         /* Note: literal invalid with % type. */
      | IDENTIFIER NUMBER  /* Note: invalid with % type. */
      ;

/* Rule section */

rules : C_IDENTIFIER rbody prec
      | rules  rule
      ;
rule  : C_IDENTIFIER rbody prec
      | '|' rbody prec
```



```
        ;
rbody : /* empty */
      | rbody IDENTIFIER
      | rbody act
        ;
act   : '{'
        {
          /* Copy action, translate $$, and so on. */
        }
        '}'
        ;
prec  : /* Empty */
      | PREC IDENTIFIER
      | PREC IDENTIFIER act
      | prec ';'
        ;
```

Conflicts

The parser produced for an input grammar may contain states in which conflicts occur. The conflicts occur because the grammar is not LALR(1). An ambiguous grammar always contains at least one LALR(1) conflict. The *yacc* utility shall resolve all conflicts, using either default rules or user-specified precedence rules.

Conflicts are either shift/reduce conflicts or reduce/reduce conflicts. A shift/reduce conflict is where, for a given state and lookahead symbol, both a shift action and a reduce action are possible. A reduce/reduce conflict is where, for a given state and lookahead symbol, reductions by two different rules are possible.

The rules below describe how to specify what actions to take when a conflict occurs. Not all shift/reduce conflicts can be successfully resolved this way because the conflict may be due to something other than ambiguity, so incautious use of these facilities can cause the language accepted by the parser to be much different from that which was intended. The description file shall contain sufficient information to understand the cause of the conflict. Where ambiguity is the reason either the default or explicit rules should be adequate to produce a working parser.

The declared precedences and associativities (see Declarations Section, on page 3616) are used to resolve parsing conflicts as follows:

1. A precedence and associativity is associated with each grammar rule; it is the precedence and associativity of the last token or literal in the body of the rule. If the **%prec** keyword is used, it overrides this default. Some grammar rules might not have both precedence and associativity.
2. If there is a shift/reduce conflict, and both the grammar rule and the input symbol have precedence and associativity associated with them, then the conflict is resolved in favor of the action (shift or reduce) associated with the higher precedence. If the precedences are the same, then the associativity is used; left associative implies reduce, right associative implies shift, and non-associative implies an error in the string being parsed.
3. When there is a shift/reduce conflict that cannot be resolved by rule 2, the shift is done. Conflicts resolved this way are counted in the diagnostic output described in Error Handling (on page 3623).

4. When there is a reduce/reduce conflict, a reduction is done by the grammar rule that occurs earlier in the input sequence. Conflicts resolved this way are counted in the diagnostic output described in Error Handling.

Conflicts resolved by precedence or associativity shall not be counted in the shift/reduce and reduce/reduce conflicts reported by *yacc* on either standard error or in the description file.

Error Handling

The token **error** shall be reserved for error handling. The name **error** can be used in grammar rules. It indicates places where the parser can recover from a syntax error. The default value of **error** shall be 256. Its value can be changed using a **%token** declaration. The lexical analyzer should not return the value of **error**.

The parser shall detect a syntax error when it is in a state where the action associated with the lookahead symbol is **error**. A semantic action can cause the parser to initiate error handling by executing the macro YYERROR. When YYERROR is executed, the semantic action passes control back to the parser. YYERROR cannot be used outside of semantic actions.

When the parser detects a syntax error, it normally calls *yyerror*( ) with the character string `"syntax error"` as its argument. The call shall not be made if the parser is still recovering from a previous error when the error is detected. The parser is considered to be recovering from a previous error until the parser has shifted over at least three normal input symbols since the last error was detected or a semantic action has executed the macro *yyerrok*. The parser shall not call *yyerror*( ) when YYERROR is executed.

The macro function YYRECOVERING shall return 1 if a syntax error has been detected and the parser has not yet fully recovered from it. Otherwise, zero shall be returned.

When a syntax error is detected by the parser, the parser shall check if a previous syntax error has been detected. If a previous error was detected, and if no normal input symbols have been shifted since the preceding error was detected, the parser checks if the lookahead symbol is an endmarker (see Interface to the Lexical Analyzer, on page 3624). If it is, the parser shall return with a non-zero value. Otherwise, the lookahead symbol shall be discarded and normal parsing shall resume.

When YYERROR is executed or when the parser detects a syntax error and no previous error has been detected, or at least one normal input symbol has been shifted since the previous error was detected, the parser shall pop back one state at a time until the parse stack is empty or the current state allows a shift over **error**. If the parser empties the parse stack, it shall return with a non-zero value. Otherwise, it shall shift over **error** and then resume normal parsing. If the parser reads a lookahead symbol before the error was detected, that symbol shall still be the lookahead symbol when parsing is resumed.

The macro *yyerrok* in a semantic action shall cause the parser to act as if it has fully recovered from any previous errors. The macro *yyclearin* shall cause the parser to discard the current lookahead token. If the current lookahead token has not yet been read, *yyclearin* shall have no effect.

The macro YYACCEPT shall cause the parser to return with the value zero. The macro YYABORT shall cause the parser to return with a non-zero value.

Interface to the Lexical Analyzer

The application shall ensure that the *yylex*( ) function is an integer-valued function that returns a *token number* greater than zero representing the kind of token read, or a value less than or equal to zero when the end of input is reached. When the parser generated by *yacc* calls *yylex*( ), it shall assign the returned value, if greater than zero, to the external variable *yychar*. If there is a value associated with the returned token (see the discussion of *tag* above), it shall be assigned to the external variable *yylval*. If the return value from *yylex*( ) is less than or equal to zero, the parser shall assign the value YYEOF to *yychar*.

If the parser and *yylex*( ) do not agree on these token numbers, reliable communication between them cannot occur. For (single-byte character) literals, the token is simply the numeric value of the character in the current character set. The numbers for other tokens can either be chosen by *yacc*, or chosen by the user. In either case, the **#define** construct of C is used to allow *yylex*( ) to return these numbers symbolically. The **#define** statements are put into the code file, and the header file if that file is requested. The set of characters permitted by *yacc* in an identifier is larger than that permitted by C. Token names found to contain such characters shall not be included in the **#define** declarations.

If the token numbers are chosen by *yacc*, the tokens other than literals shall be assigned numbers greater than 256, although no order is implied. A token can be explicitly assigned a number by following its first appearance in the declarations section with a number. Names and literals not defined this way retain their default definition. All token numbers assigned by *yacc* shall be unique and distinct from the token numbers used for literals and user-assigned tokens. If duplicate token numbers cause conflicts in parser generation, *yacc* shall report an error; otherwise, it is unspecified whether the token assignment is accepted or an error is reported.

When a parser action is executed, *yychar* shall hold either the token number of the lookahead token, or YYEMPTY indicating that there is no lookahead token, or YYEOF indicating the end of input. If *yychar* holds the token number of the lookahead token, *yylval* shall hold the value associated with that token, if any.

The application shall ensure that when the end of input is reached, the *yylex*( ) function returns a value that is zero or negative. The parser shall treat this as the token number YYEOF for a special token called the *endmarker*. If the tokens up to, but excluding, the endmarker form a structure that matches the start symbol, the parser shall accept the input. If the endmarker is seen in any other context, it shall be considered an error.

Completing the Program

In addition to *yyparse*( ) and *yylex*( ), the functions *yyerror*( ) and *main*( ) are required to make a complete program. The application can supply *main*( ) and *yyerror*( ), or those routines can be obtained from the *yacc* library.

Yacc Library

The following functions shall appear only in the *yacc* library accessible through the **–l y** operand to *c17*; they can therefore be redefined by a conforming application:

int *main*(**void**)
: This function shall call *yyparse*( ) and exit with an unspecified value. Other actions within this function are unspecified.

void *yyerror*(**const char** **s*)
: This function shall write the NUL-terminated argument to standard error, followed by a <newline>.

The order of the **–l y** and **–l l** operands given to *c17* is significant; the application shall either provide its own *main*( ) function or ensure that **–l y** precedes **–l l**.

Debugging the Parser

The parser generated by *yacc* shall have diagnostic facilities in it that can be optionally enabled at either compile time or at runtime (if enabled at compile time). The compilation of the runtime debugging code is under the control of YYDEBUG, a preprocessor symbol. If YYDEBUG has a non-zero value, the debugging code shall be included. If its value is zero, the code shall not be included.

In parsers where the debugging code has been included, the external **int** *yydebug* can be used to turn debugging on (with a non-zero value) and off (zero value) at runtime. The initial value of *yydebug* shall be zero.

When **–t** is specified, the code file shall be built such that, if YYDEBUG is not already defined at compilation time (using the *c17* **–D** YYDEBUG option, for example), YYDEBUG shall be set explicitly to 1. When **–t** is not specified, the code file shall be built such that, if YYDEBUG is not already defined, it shall be set explicitly to zero.

The format of the debugging output is unspecified but includes at least enough information to determine the shift and reduce actions, and the input symbols. It also provides information about error recovery.

Algorithms

The parser constructed by *yacc* implements an LALR(1) parsing algorithm as documented in the literature. It is unspecified whether the parser is table-driven or direct-coded.

A parser generated by *yacc* shall never request an input symbol from *yylex*( ) while in a state where the only actions other than the error action are reductions by a single rule.

The literature of parsing theory defines these concepts.

Limits

The *yacc* utility may have several internal tables. The minimum maximums for these tables are shown in the following table. The exact meaning of these values is implementation-defined. The implementation shall define the relationship between these values and between them and any error messages that the implementation may generate should it run out of space for any internal structure. An implementation may combine groups of these resources into a single pool as long as the total available to the user does not fall below the sum of the sizes specified by this section.

Table 3-23 Internal Limits in *yacc*

Limit	Minimum Maximum	Description
{NTERMS}	126	Number of tokens.
{NNONTERM}	200	Number of non-terminals.
{NPROD}	300	Number of rules.
{NSTATES}	600	Number of states.
{MEMSIZE}	5 200	Length of rules. The total length, in names (tokens and non-terminals), of all the rules of the grammar. The left-hand side is counted for each rule, even if it is not explicitly repeated, as specified in Grammar Rules in yacc (on page 3618).
{ACTSIZE}	4 000	Number of actions. ``Actions'' here (and in the description file) refer to parser actions (shift, reduce, and so on) not to semantic actions defined in Grammar Rules in yacc (on page 3618).

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

If any errors are encountered, the run is aborted and *yacc* exits with a non-zero status. Partial code files and header files may be produced. The summary information in the description file shall always be produced if the **–v** flag is present.

APPLICATION USAGE

Historical implementations experience name conflicts on the names **yacc.tmp**, **yacc.acts**, **yacc.debug**, **y.tab.c**, **y.tab.h**, and **y.output** if more than one copy of *yacc* is running in a single directory at one time. The **–b** option was added to overcome this problem. The related problem of allowing multiple *yacc* parsers to be placed in the same file was addressed by adding a **–p** option to override the previously hard-coded **yy** variable prefix.

The description of the **–p** option specifies the minimal set of function and variable names that cause conflict when multiple parsers are linked together. YYSTYPE does not need to be changed. Instead, the programmer can use **–b** to give the header files for different parsers different names, and then the file with the *yylex*( ) for a given parser can include the header for that parser. Names such as *yyclearerr* do not need to be changed because they are used only in the actions; they do not have linkage. It is possible that an implementation has other names, either internal ones for implementing things such as *yyclearerr*, or providing non-standard features that it wants to change with **–p**.

Unary operators that are the same token as a binary operator in general need their precedence adjusted. This is handled by the **%prec** advisory symbol associated with the particular grammar rule defining that unary operator. (See Grammar Rules in yacc (on page 3618).) Applications are not required to use this operator for unary operators, but the grammars that do not require it are rare.

If *yyerror*( ) and *yylex*( ) are not defined within **%{** and **%}** in the declarations section as functions or macros, nor in the programs section as functions, recommended practice is to declare them as

functions in a separate header file and include that file in the declarations section, followed by `#define yyerror yyerror` and `#define yylex yylex`. This lets the separate header file be the definitive API for all code defining or using these functions.

EXAMPLES

Access to the *yacc* library is obtained with library search operands to *c17*. To use the *yacc* library *main*():

```
c17 y.tab.c -l y
```

Both the *lex* library and the *yacc* library contain *main*(). To access the *yacc main*():

```
c17 y.tab.c lex.yy.c -l y -l l
```

This ensures that the *yacc* library is searched first, so that its *main*() is used.

The historical *yacc* libraries have contained two simple functions that are normally coded by the application programmer. These functions are similar to the following code:

```
#include <locale.h>
int main(void)
{
    extern int yyparse();

    setlocale(LC_ALL, "");

    /* If the following parser is one created by lex, the
       application must be careful to ensure that LC_CTYPE
       and LC_COLLATE are set to the POSIX locale. */
    (void) yyparse();
    return (0);
}

#include <stdio.h>

void yyerror(const char *msg)
{
    (void) fprintf(stderr, "%s\n", msg);
    return (0);
}
```

RATIONALE

The references in **Referenced Documents** may be helpful in constructing the parser generator. The referenced DeRemer and Pennello article (along with the works it references) describes a technique to generate parsers that conform to this volume of POSIX.1-2024. Work in this area continues to be done, so implementors should consult current literature before doing any new implementations. The original Knuth article is the theoretical basis for this kind of parser, but the tables it generates are impractically large for reasonable grammars and should not be used. The ``equivalent to'' wording is intentional to assure that the best tables that are LALR(1) can be generated.

There has been confusion between the class of grammars, the algorithms needed to generate parsers, and the algorithms needed to parse the languages. They are all reasonably orthogonal. In particular, a parser generator that accepts the full range of LR(1) grammars need not generate a table any more complex than one that accepts SLR(1) (a relatively weak class of LR grammars) for a grammar that happens to be SLR(1). Such an implementation need not recognize the case, either; table compression can yield the SLR(1) table (or one even smaller than that) without recognizing that the grammar is SLR(1). The speed of an LR(1) parser for any class is dependent

more upon the table representation and compression (or the code generation if a direct parser is generated) than upon the class of grammar that the table generator handles.

The speed of the parser generator is somewhat dependent upon the class of grammar it handles. However, the original Knuth article algorithms for constructing LR parsers were judged by its author to be impractically slow at that time. Although full LR is more complex than LALR(1), as computer speeds and algorithms improve, the difference (in terms of acceptable wall-clock execution time) is becoming less significant.

Potential authors are cautioned that the referenced DeRemer and Pennello article previously cited identifies a bug (an over-simplification of the computation of LALR(1) lookahead sets) in some of the LALR(1) algorithm statements that preceded it to publication. They should take the time to seek out that paper, as well as current relevant work, particularly Aho's.

The **–b** option was added to provide a portable method for permitting *yacc* to work on multiple separate parsers in the same directory. If a directory contains more than one *yacc* grammar, and both grammars are constructed at the same time (by, for example, a parallel *make* program), conflict results. While the solution is not historical practice, it corrects a known deficiency in historical implementations. Corresponding changes were made to all sections that referenced the filenames **y.tab.c** (now ``the code file''), **y.tab.h** (now ``the header file''), and **y.output** (now ``the description file'').

The grammar for *yacc* input is based on System V documentation. The textual description shows there that the `';'` is required at the end of the rule. The grammar and the implementation do not require this. (The use of **C_IDENTIFIER** causes a reduce to occur in the right place.)

Also, in that implementation, the constructs such as **%token** can be terminated by a <semicolon>, but this is not permitted by the grammar. The keywords such as **%token** can also appear in uppercase, which is again not discussed. In most places where `'%'` is used, <backslash> can be substituted, and there are alternate spellings for some of the symbols (for example, **%LEFT** can be `"%<"` or even `"\<"`).

Historically, <*tag*> can contain any characters except `'>'`, including white space, in the implementation. However, since the *tag* must reference an ISO C standard union member, in practice conforming implementations need to support only the set of characters for ISO C standard identifiers in this context.

Some historical implementations are known to accept actions that are terminated by a period. Historical implementations often allow `'$'` in names. A conforming implementation does not need to support either of these behaviors.

Deciding when to use **%prec** illustrates the difficulty in specifying the behavior of *yacc*. There may be situations in which the *grammar* is not, strictly speaking, in error, and yet *yacc* cannot interpret it unambiguously. The resolution of ambiguities in the grammar can in many instances be resolved by providing additional information, such as using **%type** or **%union** declarations. It is often easier and it usually yields a smaller parser to take this alternative when it is appropriate.

The size and execution time of a program produced without the runtime debugging code is usually smaller and slightly faster in historical implementations.

Statistics messages from several historical implementations include the following types of information:

```
n/512 terminals, n/300 non-terminals
n/600 grammar rules, n/1500 states
n shift/reduce, n reduce/reduce conflicts reported
```



```
n/350 working sets used
Memory: states, etc. n/15000, parser n/15000
n/600 distinct lookahead sets
n extra closures
n shift entries, n exceptions
n goto entries
n entries saved by goto default
Optimizer space used: input n/15000, output n/15000
n table entries, n zero
Maximum spread: n, Maximum offset: n
```

The report of internal tables in the description file is left implementation-defined because all aspects of these limits are also implementation-defined. Some implementations may use dynamic allocation techniques and have no specific limit values to report.

The format of the **y.output** file is not given because specification of the format was not seen to enhance applications portability. The listing is primarily intended to help human users understand and debug the parser; use of **y.output** by a conforming application script would be unusual. Furthermore, implementations have not produced consistent output and no popular format was apparent. The format selected by the implementation should be human-readable, in addition to the requirement that it be a text file.

Standard error reports are not specifically described because they are seldom of use to conforming applications and there was no reason to restrict implementations.

Some implementations recognize `"={"` as equivalent to `'{'` because it appears in historical documentation. This construction was recognized and documented as obsolete as long ago as 1978, in the referenced *Yacc: Yet Another Compiler-Compiler*. This volume of POSIX.1-2024 chose to leave it as obsolete and omit it.

Multi-byte characters should be recognized by the lexical analyzer and returned as tokens. They should not be returned as multi-byte character literals. The token **error** that is used for error recovery is normally assigned the value 256 in the historical implementation. Thus, the token value 256, which is used in many multi-byte character sets, is not available for use as the value of a user-defined token.

Earlier versions of this standard did not require the code file created by *yacc* to contain declarations of *yyerror*(), *yylex*(), and *yyparse*(). This meant that portable applications that did not define them had to declare them in the *grammar* file, to ensure they would not be diagnosed by the compiler as being called without being declared, but this was not stated in those versions of the standard either. The standard developers decided it was preferable for *yacc* to include the declarations in the code file and this is now a requirement. However, the declarations of *yyerror*() and *yylex*() are only visible if a macro of the same name is not defined, which provides application writers with a way to suppress the declaration if desired (for example, in order to provide their own declaration that would conflict with the one written by *yacc*). These functions are not declared in the header file because a macro definition in the declaration section would not be be able to suppress them there.

Earlier versions of this standard were also silent about a declaration of *main*(). However, the equivalent solution was not adopted because a declaration of *main*() would only be needed if it is called recursively by an application. Although in theory an application could call the *yacc* library version of *main*() from code in a *grammar* file, it is questionable why any application (other than a test suite) would do so, in particular because that version of *main*() does not accept any arguments and it calls *exit*()—it does not return—and therefore is of little use recursively. An application that includes its own definition of *main*() could call it recursively, but can

reasonably be expected to ensure it does not call *main*( ) without previously defining or declaring it. An additional complication is that *main*( ) has multiple different allowed prototypes. The standard developers decided the simplest solution was to disallow *yacc* from providing a declaration of *main*( ) in the code file.

FUTURE DIRECTIONS

If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.

SEE ALSO

c17, *lex*

XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

This utility is marked as part of the C-Language Development Utilities option.

Minor changes have been added to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ``must'' for application requirements.

IEEE PASC Interpretation 1003.2 #177 is applied, changing the comment on **RCURL** from the }% token to the %}.

Issue 7

Austin Group Interpretation 1003.1-2001 #190 is applied, clarifying the requirements for generated code to conform to the ISO C standard.

Austin Group Interpretation 1003.1-2001 #191 is applied, clarifying the handling of C-language trigraphs and curly brace preprocessing tokens.

SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0204 [977] is applied.

Issue 8

Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

Austin Group Defect 1269 is applied, changing the required contents of the code file (including **#define** statements for YYEMPTY and YYEOF) and adding new requirements for the interface to the lexical analyzer.

Austin Group Defect 1388 is applied, changing the requirements relating to declarations of *yyerror*( ), *yylex*( ), *yyparse*( ), and *main*( ).

NAME

zcat — expand and concatenate data

SYNOPSIS

XSI

```
zcat [file...]
```

DESCRIPTION

Refer to *compress*.

124410 *The Open Group Standard*

124411 **Vol. 4:**

124412 **Rationale (Informative), Issue 8**

124413 *The Open Group*
124414 *The Institute of Electrical and Electronics Engineers, Inc.*

Rationale (Informative)

Part A:

Base Definitions

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

124420 *Appendix A*

124421 # *Rationale for Base Definitions*

124422 ## A.1 Introduction

124423 ### A.1.1 Scope

124424 POSIX.1-2024 is one of a family of standards known as POSIX. The family of standards extends
124425 to many topics; POSIX.1 consists of both operating system interfaces and shell and utilities.
124426 POSIX.1-2024 is technically identical to The Open Group Base Specifications, Issue 8.

124427 **Scope of POSIX.1-2024**

124428 The (paraphrased) goals of this development were to revise the single document that is ISO/IEC
124429 9945:2009 Parts 1 through 4 as amended by ISO/IEC 9945:2009/Cor.1:2013 and ISO/IEC
124430 9945:2009/Cor.2:2017, IEEE Std 1003.1-2017, and the appropriate parts of The Open Group
124431 Single UNIX Specification, Version 5. This work has been undertaken by the Austin Group, a
124432 joint working group of IEEE, The Open Group, and ISO/IEC JTC 1/SC 22.

124433 The following are the base documents in this version:

124434 • IEEE Std 1003.1-2017

124435 • IEEE Std 1003.26-2003

124436 • ISO/IEC 9899: 2018, Programming Languages — C

124437 This version has addressed the following areas:

124438 • Issues raised by Austin Group defect reports, IEEE Interpretations against IEEE Std 1003.1,
124439 and ISO/IEC defect reports against ISO/IEC 9945

124440 The repository of interpretations can be accessed at www.opengroup.org/austin/interps.

124441 • Issues raised in corrigenda for The Open Group Standards and working group resolutions
124442 from The Open Group

124443 • Changes to make the text self-consistent with the additional material merged

124444 A list of the new interfaces is included in Section B.1.1 (on page 3731).

124445 • Features, marked obsolescent in the base documents, have been considered for removal in
124446 this version

124447 See Section B.1.1 (on page 3731) and Section C.1.1 (on page 3855).

124448 • Alignment with the ISO/IEC 9899: 2018 standard

124449 The following were requirements on POSIX.1-2024:

124450 • Backward-compatibility

124451 For interfaces carried forward, it was agreed that there should be no breakage of
124452 functionality in the existing base documents. All strictly conforming applications will be

conforming but not necessarily strictly conforming to the revised standard. The goal is for system implementations to be able to support the existing and revised standards simultaneously.

- Architecture and *n*-bit-neutral

  The common standard should not make any implicit assumptions about the system architecture or size of data types; for example, previously some 32-bit implicit assumptions had crept into the standards.

- Extensibility

  It should be possible to extend the common standard without breaking backwards-compatibility; for example, the name space should be reserved and structured to avoid duplication of names between the standard and extensions to it.

POSIX.1 and the ISO C Standard

The standard developers believed it essential for a programmer to have a single complete reference place, but recognized that deference to the formal standard has to be addressed for the duplicate interface definitions between the ISO C standard and POSIX.1-2024.

Where an interface has a version in the ISO C standard, the DESCRIPTION section describes the relationship to the ISO C standard and markings are included as appropriate to show where the ISO C standard has been extended in the text.

A block of text is included at the start of each affected reference page stating whether the page is aligned with the ISO C standard or extended. Each page has been parsed for additions beyond the ISO C standard (that is, including both POSIX and UNIX extensions), and these extensions are marked as CX extensions (for C extensions).

FIPS Requirements

The Federal Information Processing Standards (FIPS) are a series of US government procurement standards managed and maintained on behalf of the US Department of Commerce by the National Institute of Standards and Technology (NIST).

The following restrictions were integrated into IEEE Std 1003.1-2001. They originally came from FIPS 151-2 which was withdrawn by NIST on February 25 2000.

- The implementation supports _POSIX_CHOWN_RESTRICTED.
- The limit {NGROUPS_MAX} is greater than or equal to 8.
- The implementation supports the setting of the group ID of a file (when it is created) to that of the parent directory.
- The implementation supports _POSIX_SAVED_IDS.
- The implementation supports _POSIX_VDISABLE.
- The implementation supports _POSIX_JOB_CONTROL.
- The implementation supports _POSIX_NO_TRUNC.
- The *read*( ) function returns the number of bytes read when interrupted by a signal and does not return –1.
- The *write*( ) function returns the number of bytes written when interrupted by a signal and does not return –1.

- In the environment for the login shell, the environment variables *LOGNAME* and *HOME* are defined and have the properties described in POSIX.1-2024.
- The value of {CHILD_MAX} is greater than or equal to 25.
- The value of {OPEN_MAX} is greater than or equal to 20.
- The implementation supports the functionality associated with the symbols CS7, CS8, CSTOPB, PARODD, and PARENB defined in **<termios.h>**.

A.1.2 Word Usage

The content of this section is mandated by IEEE and consequently it cannot be combined with the ``Terminology'' section.

Note that where the footnotes state that ``must'' is used only to describe unavoidable situations and ``will'' is only used in statements of fact, they are referring to uses of these words in normative text. In informative text, they are used in other ways with their usual dictionary meanings.

A.1.3 Conformance

See Section A.2 (on page 3643).

A.1.4 Normative References

There is no additional rationale provided for this section.

A.1.5 Change History

For Issue 7 onwards, in references to Technical Corrigenda, the original Austin Group defect report numbers that gave rise to the change are included in square brackets after the change number from the Technical Corrigendum. For more information on Austin Group defect reports see www.opengroup.org/austin/defectform.html.

A.1.6 Terminology

The meanings specified in POSIX.1-2024 for the words *shall*, *should*, and *may* are mandated by ISO/IEC directives.

In the Rationale (Informative) volume of POSIX.1-2024, the words *shall*, *should*, and *may* are sometimes used to illustrate similar usages in POSIX.1-2024. However, the rationale itself does not specify anything regarding implementations or applications.

conformance document

As a practical matter, the conformance document is effectively part of the system documentation. Conformance documents are distinguished by POSIX.1-2024 so that they can be referred to distinctly.

implementation-defined

This definition is analogous to that of the ISO C standard and, together with ``undefined'' and ``unspecified'', provides a range of specification of freedom allowed to the interface implementor.

may

The use of *may* has been limited as much as possible, due both to confusion stemming from its ordinary English meaning and to objections regarding the desirability of having as few options as possible and those as clearly specified as possible.

The usage of *can* and *may* were selected to contrast optional application behavior (can) against optional implementation behavior (may).

shall

Declarative sentences are sometimes used in POSIX.1-2024 as if they included the word *shall*, and facilities thus specified are no less required. For example, the two statements:

1. The *foo*( ) function shall return zero.
2. The *foo*( ) function returns zero.

are meant to be exactly equivalent.

should

In POSIX.1-2024, the word *should* does not usually apply to the implementation, but rather to the application. Thus, the important words regarding implementations are *shall*, which indicates requirements, and *may*, which indicates options.

obsolescent

The term ``obsolescent'' means ``do not use this feature in new applications''. A feature noted as obsolescent is supported by all implementations, but may be removed in a future version; new applications should not use these features. The obsolescence concept is not an ideal solution, but was used as a method of increasing consensus: many more objections would be heard from the user community if some of these historical features were suddenly removed without the grace period obsolescence implies. The phrase ``may be removed in a future version'' implies that the result of that consideration might in fact keep those features indefinitely if the predominance of applications do not migrate away from them quickly.

legacy

The term ``legacy'' was included in earlier versions of this standard but is no longer used in the current version.

system documentation

The system documentation should normally describe the whole of the implementation, including any extensions provided by the implementation. Such documents normally contain information at least as detailed as the specifications in POSIX.1-2024. Few requirements are made on the system documentation, but the term is needed to avoid a dangling pointer where the conformance document is permitted to point to the system documentation.

undefined

See *implementation-defined*.

unspecified

See *implementation-defined*.

The definitions for ``unspecified'' and ``undefined'' appear nearly identical at first examination, but are not. The term ``unspecified'' means that a conforming application may deal with the unspecified behavior, and it should not care what the outcome is. The term ``undefined'' says that a conforming application should not do it because no definition is provided for what it does (and implicitly it would care what the outcome was if it tried it). It is important to remember, however, that if the syntax permits the statement at all, it must have some outcome in a real implementation.

Thus, the terms ``undefined'' and ``unspecified'' apply to the way the application should think about the feature. In terms of the implementation, it is always ``defined''—there is always some result, even if it is an error. The implementation is free to choose the behavior it prefers.

This also implies that an implementation, or another standard, could specify or define the result in a useful fashion. The terms apply to POSIX.1-2024 specifically.

The term ``implementation-defined'' implies requirements for documentation that are not required for ``undefined'' (or ``unspecified''). Where there is no need for a conforming program to know the definition, the term ``undefined'' is used, even though ``implementation-defined'' could also have been used in this context. There could be a fourth term, specifying ``this standard does not say what this does; it is acceptable to define it in an implementation, but it does not need to be documented'', and undefined would then be used very rarely for the few things for which any definition is not useful. In particular, implementation-defined is used where it is believed that certain classes of application will need to know such details to determine whether the application can be successfully ported to the implementation. Such applications are not always strictly portable, but nevertheless are common and useful; often the requirements met by the application cannot be met without dealing with the issues implied by ``implementation-defined''. In some places the text refers to facilities supplied by the implementation that are outside the standard as implementation-supplied or implementation-provided. This is not intended to imply a requirement for documentation. If it were, the term ``implementation-defined'' would have been used.

In many places POSIX.1-2024 is silent about the behavior of some possible construct. For example, a variable may be defined for a specified range of values and behaviors are described for those values; nothing is said about what happens if the variable has any other value. That kind of silence can imply an error in the standard, but it may also imply that the standard was intentionally silent and that any behavior is permitted. There is a natural tendency to infer that if the standard is silent, a behavior is prohibited. That is not the intent. Silence is intended to be equivalent to the term ``unspecified''.

Three terms used within POSIX.1-2024 overlap in meaning: ``macro'', ``symbolic name'', and ``symbolic constant''.

macro
This usually describes a C preprocessor symbol, the result of the **#define** operator, with or without an argument. It may also be used to describe similar mechanisms in editors and text processors.

symbolic name
In earlier versions of this standard this was also sometimes used to refer to a C preprocessor symbol (without arguments), but the intention is for all such uses to have been removed. It is now mainly used to refer to the names for characters in character sets, but is sometimes used to refer to host names and even filenames.

symbolic constant
This also refers to a C preprocessor symbol, with specific associated requirements. See the definition in Section 3.363 (on page 84).

A.1.7 Definitions and Concepts

There is no additional rationale provided for this section.

A.1.8 Portability

To aid the identification of options within POSIX.1-2024, a notation consisting of margin codes and shading is used. This is based on the notation used in earlier versions of The Open Group Base specifications.

The benefit of this approach is a reduction in the number of *if* statements within the running text, that makes the text easier to read, and also an identification to the programmer that they need to ensure that their target platforms support the underlying options. For example, if functionality is marked with RPP in the margin, it will be available on all systems supporting the Robust Mutex Priority Protection option, but may not be available on some others.

A.1.8.1 Codes

This section includes codes for options defined in XBD Section 2.1.6 (on page 25), and the following additional codes for other purposes:

CX This margin code is used to denote extensions beyond and, in exceptional cases, deviations from the ISO C standard. For interfaces that are duplicated between POSIX.1-2024 and the ISO C standard, a CX introduction block describes the nature of the duplication, with any extensions or deviations appropriately CX marked and shaded. Where deviations exist, the reasons for them are explained in the RATIONALE section of the affected interface. Deviations have become necessary because there is no longer any formal way for ISO to acknowledge defects in the ISO C standard. For the original C90 standard and the C99 revision, defect reports (DRs) were issued, but there is no equivalent mechanism for the current revision. Even if the defect is corrected in a later revision, without stating deviations POSIX.1-2024 would continue to require the incorrect behavior described in the version of the ISO C standard that it references.

Where an interface is added to an ISO C standard header, within the header the interface has an appropriate margin marker and shading (for example, CX, XSI, TSF, and so on) and the same marking appears on the reference page in the SYNOPSIS section. This enables a programmer to easily identify that the interface is extending an ISO C standard header.

Austin Group Defect 1755 is applied, changing the CX code description to include intentional conflicts (deviations).

MX and MXX
These two margin codes both relate to the IEC 60559 Floating-Point option. The MX code denotes functionality that is mandated by the ISO C standard for IEC 60559 implementations; the MXX code denotes IEC 60559 functionality that is an extension to the ISO C standard.

MXC This margin code is used to denote functionality related to the IEC 60559 Complex Floating-Point option.

OB This margin code is used to denote obsolescent behavior and thus flag a possible future applications portability warning.

124659 OH The Single UNIX Specification has historically tried to reduce the number of headers an
124660 application has had to include when using a particular interface. Sometimes this was
124661 fewer than the base standard, and hence a notation is used to flag which headers are
124662 optional if you are using a system supporting the XSI option.

124663 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0001 [591] is applied.

124664 *A.1.8.2 Margin Code Notation*

124665 Since some features may depend on one or more options, or require more than one option, a
124666 notation is used. Where a feature requires support of a single option, a single margin code will
124667 occur in the margin. If it depends on two options and both are required, then the codes will
124668 appear with a <space> separator. If either of two options are required, then a logical OR is
124669 denoted using the '|' symbol. If more than two codes are used, a special notation is used.

124670 A.2 Conformance

124671 The terms ``profile'' and ``profiling'' are used throughout this section.

124672 A profile of a standard or standards is a codified set of option selections, such that by being
124673 conformant to a profile, particular classes of users are specifically supported.

124674 A.2.1 Implementation Conformance

124675 These definitions allow application developers to know what to depend on in an
124676 implementation.

124677 There is no definition of a ``strictly conforming implementation''; that would be an
124678 implementation that provides *only* those facilities specified by POSIX.1 with no extensions
124679 whatsoever. This is because no actual operating system implementation can exist without
124680 system administration and initialization facilities that are beyond the scope of POSIX.1.

124681 *A.2.1.1 Requirements*

124682 The word ``support'' is used in certain instances, rather than ``provide'', in order to allow an
124683 implementation that has no resident software development facilities, but that supports the
124684 execution of a *Strictly Conforming POSIX.1 Application*, to be a *conforming implementation*.

124685 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0002 [810] is applied.

124686 *A.2.1.2 Documentation*

124687 The conformance documentation is required to use the same numbering scheme as POSIX.1 for
124688 purposes of cross-referencing. All options that an implementation chooses are reflected in
124689 **<limits.h>** and **<unistd.h>**.

124690 Note that the use of ``may'' in terms of where conformance documents record where
124691 implementations may vary, implies that it is not required to describe those features identified as
124692 undefined or unspecified.

124693 Other aspects of systems must be evaluated by purchasers for suitability. Many systems
124694 incorporate buffering facilities, maintaining updated data in volatile storage and transferring
124695 such updates to non-volatile storage asynchronously. Various exception conditions, such as a
124696 power failure or a system crash, can cause this data to be lost. The data may be associated with a

file that is still open, with one that has been closed, with a directory, or with any other internal system data structures associated with permanent storage. This data can be lost, in whole or part, so that only careful inspection of file contents could determine that an update did not occur.

Also, interrelated file activities, where multiple files and/or directories are updated, or where space is allocated or released in the file system structures, can leave inconsistencies in the relationship between data in the various files and directories, or in the file system itself. Such inconsistencies can break applications that expect updates to occur in a specific sequence, so that updates in one place correspond with related updates in another place.

For example, if a user creates a file, places information in the file, and then records this action in another file, a system or power failure at this point followed by restart may result in a state in which the record of the action is permanently recorded, but the file created (or some of its information) has been lost. The consequences of this to the user may be undesirable. For a user on such a system, the only safe action may be to require the system administrator to have a policy that requires, after any system or power failure, that the entire file system must be restored from the most recent backup copy (causing all intervening work to be lost).

The characteristics of each implementation will vary in this respect and may or may not meet the requirements of a given application or user. Enforcement of such requirements is beyond the scope of POSIX.1. It is up to the purchaser to determine what facilities are provided in an implementation that affect the exposure to possible data or sequence loss, and also what underlying implementation techniques and/or facilities are provided that reduce or limit such loss or its consequences.

A.2.1.3 POSIX Conformance

This really means conformance to the base standard; however, since this document includes the core material of the Single UNIX Specification, the standard developers decided that it was appropriate to segment the conformance requirements into two, the former for the base standard, and the latter for the Single UNIX Specification (denoted XSI Conformance).

Within POSIX.1 there are some symbolic constants that, if defined to a certain value or range of values, indicate that a certain option is enabled. Other symbolic constants exist in POSIX.1 for other reasons.

In this version, some features that were previously optional have been made mandatory. For backwards compatibility, the symbolic constants associated with the option are still required now with fixed allowable ranges or values. The following options from previous versions of this standard are now mandatory:

124731 _POSIX_ASYNCHRONOUS_IO
124732 _POSIX_BARRIERS
124733 _POSIX_CLOCK_SELECTION
124734 _POSIX_MAPPED_FILES
124735 _POSIX_MEMORY_PROTECTION
124736 _POSIX_MONOTONIC_CLOCK
124737 _POSIX_READER_WRITER_LOCKS
124738 _POSIX_REALTIME_SIGNALS
124739 _POSIX_SEMAPHORES
124740 _POSIX_SPIN_LOCKS
124741 _POSIX_THREAD_SAFE_FUNCTIONS
124742 _POSIX_THREADS
124743 _POSIX_TIMEOUTS
124744 _POSIX_TIMERS

124745 A POSIX-conformant system may support the XSI option required by the Single UNIX
124746 Specification. This was intentional since the standard developers intend them to be upwards-
124747 compatible, so that a system conforming to the Single UNIX Specification can also conform to
124748 the base standard at the same time.

124749 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0003 [637] is applied.

124750 Austin Group Defect 729 is applied, adding _POSIX_DEVICE_CONTROL.

124751 Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.

124752 *A.2.1.4 XSI Conformance*

124753 This section is included to describe the conformance requirements for the base volumes of the
124754 Single UNIX Specification.

124755 XSI conformance can be thought of as a profile, selecting certain options from POSIX.1-2024.

124756 *A.2.1.5 Option Groups*

124757 The concept of ``Option Groups'' is included to allow collections of related functions or options
124758 to be grouped together. This has been used as follows: the ``XSI Option Groups'' have been
124759 created to allow super-options, collections of underlying options and related functions, to be
124760 collectively supported by XSI-conforming systems.

124761 The standard developers considered the matter of subprofiling and decided it was better to
124762 include an enabling mechanism rather than detailed normative requirements. A set of
124763 subprofiling options was developed and included later in this volume of POSIX.1-2024 as an
124764 informative illustration.

124765 **Subprofiling Considerations**

124766 The goal of not simultaneously fixing maximums and minimums was to allow implementations
124767 of the base standard or standards to support multiple profiles without conflict.

The following summarizes the rules for the limit types:

Limit Type	Fixed Value	Minimum Acceptable Value	Maximum Acceptable Value
Standard Profile	*Xs* *Xp* == *Xs* (No change)	*Ys* *Yp* >= *Ys* (May increase the limit)	*Zs* *Zp* <= *Zs* (May decrease the limit)

The intent is that ranges specified by limits in profiles be entirely contained within the corresponding ranges of the base standard or standards being profiled, and that the unlimited end of a range in a base standard must remain unlimited in any profile of that standard.

Thus, the fixed _POSIX_* limits are constants and must not be changed by a profile. The variable counterparts (typically without the leading _POSIX_) can be changed but still remain semantically the same; that is, they still allow implementation values to vary as long as they meet the requirements for that value (be it a minimum or maximum).

Where a profile does not provide a feature upon which a limit is based, the limit is not relevant. Applications written to that profile should be written to operate independently of the value of the limit.

An example which has previously allowed implementations to support both the base standard and two other profiles in a compatible manner follows:

```
Base standard (POSIX.1-1996): _POSIX_CHILD_MAX 6
Base standard: CHILD_MAX   minimum maximum _POSIX_CHILD_MAX
    FIPS profile/SUSv2  CHILD_MAX   25 (minimum maximum)
```

Another example:

```
Base standard (POSIX.1-1996): _POSIX_NGROUPS_MAX 0
Base standard: NGROUPS_MAX   minimum maximum _POSIX_NGROUP_MAX
    FIPS profile/SUSv2  NGROUPS_MAX   8
```

A profile may lower a minimum maximum below the equivalent _POSIX value:

```
Base standard: _POSIX_foo_MAX   Z
Base standard: foo_MAX   _POSIX_foo_MAX
    profile standard : foo_MAX   X  (X can be less than, equal to,
                                     or greater than  _POSIX_foo_MAX)
```

In this case an implementation conforming to the profile may not conform to the base standard, but an implementation to the base standard will conform to the profile.

XSI Option Groups

Austin Group Defect 1192 is applied, marking the *encrypt*() and *setkey*() functions as obsolescent.

Austin Group Defect 1346 is applied, removing _POSIX_MONOTONIC_CLOCK from the Advanced Realtime option group.

A.2.1.6 Options

The final subsections within *Implementation Conformance* list the core options within POSIX.1-2024. This includes both options for the System Interfaces volume of POSIX.1-2024 and the Shell and Utilities volume of POSIX.1-2024.

Austin Group Defect 190 is applied, adding *man* to the list of utilities in the User Portability

Utilities option.

A.2.2 Application Conformance

These definitions guide users or adapters of applications in determining on which implementations an application will run and how much adaptation would be required to make it run on others. These definitions are modeled after related ones in the ISO C standard.

POSIX.1 occasionally uses the expressions ``portable application'' or ``conforming application''. As they are used, these are synonyms for any of these terms. The differences between the classes of application conformance relate to the requirements for other standards, the options supported (such as the XSI option) or, in the case of the Conforming POSIX.1 Application Using Extensions, to implementation extensions. When one of the less explicit expressions is used, it should be apparent from the context of the discussion which of the more explicit names is appropriate

A.2.2.1 Strictly Conforming POSIX Application

This definition is analogous to that of an ISO C standard ``conforming program''.

The major difference between a Strictly Conforming POSIX Application and an ISO C standard strictly conforming program is that the latter is not allowed to use features of POSIX that are not in the ISO C standard.

A.2.2.2 Conforming POSIX Application

Examples of <National Bodies> include ANSI, BSI, and AFNOR.

A.2.2.3 Conforming POSIX Application Using Extensions

Due to possible requirements for configuration or implementation characteristics in excess of the specifications in **<limits.h>** or related to the hardware (such as array size or file space), not every Conforming POSIX Application Using Extensions will run on every conforming implementation.

A.2.2.4 Strictly Conforming XSI Application

This is intended to be upwards-compatible with the definition of a Strictly Conforming POSIX Application, with the addition of the facilities and functionality included in the XSI option.

A.2.2.5 Conforming XSI Application Using Extensions

Such applications may use extensions beyond the facilities defined by POSIX.1-2024 including the XSI option, but need to document the additional requirements.

A.2.3 Language-Dependent Services for the C Programming Language

POSIX.1 is, for historical reasons, both a specification of an operating system interface, shell and utilities, and a C binding for that specification. Efforts had been previously undertaken to generate a language-independent specification; however, that had failed, and the fact that the ISO C standard is the *de facto* primary language on POSIX and the UNIX system makes this a necessary and workable situation.

A.2.4 Other Language-Related Specifications

There is no additional rationale provided for this section.

A.3 Definitions

The definitions in this section are stated so that they can be used as exact substitutes for the terms in text. They should not contain requirements or cross-references to sections within POSIX.1-2024; that is accomplished by using an informative note. In addition, the term should not be included in its own definition. Where requirements or descriptions need to be addressed but cannot be included in the definitions, due to not meeting the above criteria, these occur in the General Concepts chapter.

In this version, the definitions have been reworked extensively to meet style requirements and to include terms from the base documents (see the Scope).

Many of these definitions are necessarily circular, and some of the terms (such as ``process'') are variants of basic computing science terms that are inherently hard to define. Where some definitions are more conceptual and contain requirements, these appear in the General Concepts chapter. Those listed in this section appear in an alphabetical glossary format of terms.

Some definitions must allow extension to cover terms or facilities that are not explicitly mentioned in POSIX.1-2024. For example, the definition of ``Extended Security Controls'' permits implementations beyond those defined in POSIX.1-2024.

Some terms in the following list of notes do not appear in POSIX.1-2024; these are marked suffixed with an asterisk (*). Many of them have been specifically excluded from POSIX.1-2024 because they concern system administration, implementation, or other issues that are not specific to the programming interface. Those are marked with a reason, such as ``implementation-defined''.

Alias Name

Austin Group Defect 1050 is applied, adding `'-'` to the characters that can be used in an alias name.

Anonymous Memory Object

Austin Group Defect 850 is applied, adding anonymous memory objects.

Application

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0004 [937] is applied.

Appropriate Privileges

One of the fundamental security problems with many historical UNIX systems has been that the privilege mechanism is monolithic—a user has either no privileges or *all* privileges. Thus, a successful ``trojan horse'' attack on a privileged process defeats all security provisions. Therefore, POSIX.1 allows more granular privilege mechanisms to be defined. For many historical implementations of the UNIX system, the presence of the term ``appropriate privileges'' in POSIX.1 may be understood as a synonym for ``superuser'' (UID 0). However, other systems have emerged where this is not the case and each discrete controllable action has *appropriate privileges* associated with it. Because this mechanism is implementation-defined, it must be described in the conformance document. Although that description affects several parts of POSIX.1 where the term ``appropriate privilege'' is used, because the term ``implementation-defined'' only appears here, the description of the entire mechanism and its effects on these other sections belongs in this equivalent section of the conformance document. This is especially convenient for implementations with a single mechanism that applies in all areas, since it only needs to be described once.

Async-Signal-Safe Function

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0005 [516] is applied.

Background Job

Austin Group Defect 1254 is applied, changing this definition.

Base Character*

The term ``Base Character'' has been removed, as it was felt that the use of this term within POSIX.1-2024 was common usage English.

Basename

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0006 [653] is applied.

Built-In Utility

Austin Group Defect 854 is applied, changing text relating to regular built-in utilities.

Byte

The restriction that a byte is now exactly eight bits was a conscious decision by the standard developers. It came about due to a combination of factors, primarily the use of the type **int8_t** within the networking functions and the alignment with the ISO/IEC 9899:1999 standard, where the **intN_t** types were first defined.

According to the ISO/IEC 9899:1999 standard:

- The **[u]intN_t** types must be two's complement with no padding bits and no illegal values.
- All types (apart from bit fields, which are not relevant here) must occupy an integral number of bytes.
- If a type with width W occupies B bytes with C bits per byte (C is the value of {CHAR_BIT}), then it has P padding bits where $P+W=B*C$.

- Therefore, for **int8_t** P=0, W=8. Since $B \geq 1$, $C \geq 8$, the only solution is B=1, C=8.

The standard developers also felt that this was not an undue restriction for the current state-of-the-art for this version of the standard, but recognize that if industry trends continue, a wider character type may be required in the future.

Character

The term ``character'' is used to mean a sequence of one or more bytes representing a member of a character set. The deviation in the exact text of the ISO C standard definition for ``byte'' meets the intent of the rationale of the ISO C standard also clears up the ambiguity raised by the term ``basic execution character set''. The octet-minimum requirement is a reflection of the {CHAR_BIT} value.

Austin Group Defect 1356 is applied, changing the definition of ``character'' to match the definition of the term ``multi-byte character'' in the ISO C standard.

Child Process

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/3 is applied, adding the *vfork*() function to those listed.

Clock Tick

The ISO C standard defines a similar interval for use by the *clock*() function. There is no requirement that these intervals be the same. In historical implementations these intervals are different.

Code Block

Austin Group Defect 613 is applied, adding this definition.

Command

The terms ``command'' and ``utility'' are related but have distinct meanings. Command is defined as ``a directive to a shell to perform a specific task''. The directive can be in the form of a single utility name (for example, *ls*), or the directive can take the form of a compound command (for example, `"ls | grep name | pr"`). A utility is a program that can be called by name from a shell. Issuing only the name of the utility to a shell is the equivalent of a one-word command. A utility may be invoked as a separate program that executes in a different process than the command language interpreter, or it may be implemented as a part of the command language interpreter. For example, the *echo* command (the directive to perform a specific task) may be implemented such that the *echo* utility (the logic that performs the task of echoing) is in a separate program; therefore, it is executed in a process that is different from the command language interpreter. Conversely, the logic that performs the *echo* utility could be built into the command language interpreter; therefore, it could execute in the same process as the command language interpreter.

The terms ``tool'' and ``application'' can be thought of as being synonymous with ``utility'' from the perspective of the operating system kernel. Tools, applications, and utilities historically have run, typically, in processes above the kernel level. Tools and utilities historically have been a part of the operating system non-kernel code and have performed system-related functions, such as listing directory contents, checking file systems, repairing file systems, or extracting system status information. Applications have not generally been a part of the operating system, and they perform non-system-related functions, such as word processing, architectural design, mechanical design, workstation publishing, or financial analysis. Utilities have most frequently been provided by the operating system distributor, applications by third-party software

distributors, or by the users themselves. Nevertheless, POSIX.1-2024 does not differentiate between tools, utilities, and applications when it comes to receiving services from the system, a shell, or the standard utilities. (For example, the *xargs* utility invokes another utility; it would be of fairly limited usefulness if the users could not run their own applications in place of the standard utilities.) Utilities are not applications in the sense that they are not themselves subject to the restrictions of POSIX.1-2024 or any other standard—there is no requirement for *grep*, *stty*, or any of the utilities defined here to be any of the classes of conforming applications.

Column Positions

In most 1-byte character sets, such as ASCII, the concept of column positions is identical to character positions and to bytes. Therefore, it has been historically acceptable for some implementations to describe line folding or tab stops or table column alignment in terms of bytes or character positions. Other character sets pose complications, as they can have internal representations longer than one octet and they can have display characters that have different widths on the terminal screen or printer.

In POSIX.1-2024 the term ``column positions'' has been defined to mean character—not byte—positions in input files. Output files describe the column position in terms of the display width of the narrowest printable character in the character set, adjusted to fit the characteristics of the output device. It is very possible that *n* column positions will not be able to hold *n* characters in some character sets, unless all of those characters are of the narrowest width. It is assumed that the implementation is aware of the width of the various characters, deriving this information from the value of *LC_CTYPE*, and thus can determine how many column positions to allot for each character in those utilities where it is important.

The term ``column position'' was used instead of the more natural ``column'' because the latter is frequently used in the different contexts of columns of figures, columns of table values, and so on. Wherever confusion might result, these latter types of columns are referred to as ``text columns''.

Condition Variable

Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899: 2018 standard.

Control Operator

Austin Group Defect 449 is applied, adding **;** & to the list of control operators.

Controlling Terminal

The question of which of possibly several special files referring to the terminal is meant is not addressed in POSIX.1. The pathname **/dev/tty** is a synonym for the controlling terminal associated with a process.

Core Image

Austin Group Defect 1141 is applied, replacing the core file definition with a core image definition.

CPU Time (Execution Time)

Austin Group Defect 1116 is applied, removing a reference to the Threads option that existed in earlier versions of this standard.

Decimal-Point Character

Austin Group Defect 1449 is applied, adding this definition.

Declaration Utility

Austin Group Defect 351 is applied, adding this definition.

Device Number*

The concept is handled in *stat*( ) as *ID of device*.

Direct I/O

Historically, direct I/O refers to the system bypassing intermediate buffering, but may be extended to cover implementation-defined optimizations.

Directory

The format of the directory file is implementation-defined and differs radically between System V and 4.3 BSD. However, routines (derived from 4.3 BSD) for accessing directories and certain constraints on the format of the information returned by those routines are described in the **<dirent.h>** header.

Directory Entry (or Hard Link)

Austin Group Defect 1380 is applied, changing ``link'' to ``hard link''.

Display

The Shell and Utilities volume of POSIX.1-2024 assigns precise requirements for the terms ``display'' and ``write''. Some historical systems have chosen to implement certain utilities without using the traditional file descriptor model. For example, the *vi* editor might employ direct screen memory updates on a personal computer, rather than a *write*( ) system call. An instance of user prompting might appear in a dialog box, rather than with standard error. When the Shell and Utilities volume of POSIX.1-2024 uses the term ``display'', the method of outputting to the terminal is unspecified; many historical implementations use *termcap* or *terminfo*, but this is not a requirement. The term ``write'' is used when the Shell and Utilities volume of POSIX.1-2024 mandates that a file descriptor be used and that the output can be redirected. However, it is assumed that when the writing is directly to the terminal (it has not been redirected elsewhere), there is no practical way for a user or test suite to determine whether a file descriptor is being used. Therefore, the use of a file descriptor is mandated only for the redirection case and the implementation is free to use any method when the output is not redirected. The verb *write* is used almost exclusively, with the very few exceptions of those utilities where output redirection need not be supported: *tabs*, *talk*, *tput*, and *vi*.

125028 **Dot**

125029 The symbolic name *dot* is carefully used in POSIX.1 to distinguish the working directory
125030 filename from a period or a decimal point.

125031 **Dot-Dot**

125032 Historical implementations permit the use of these filenames without their special meanings.
125033 Such use precludes any meaningful use of these filenames by a Conforming POSIX.1
125034 Application. Therefore, such use is considered an extension, the use of which makes an
125035 implementation non-conforming; see also Section A.4.16 (on page 3683).

125036 **Dot-Po File**

125037 Austin Group Defect 1122 is applied, adding this definition.

125038 **Empty Directory**

125039 Austin Group Defect 1380 is applied, changing ``link'' to ``hard link''.

125040 **Epoch**

125041 Historically, the origin of UNIX system time was referred to as ``00:00:00 GMT, January 1, 1970''.
125042 Greenwich Mean Time is actually not a term acknowledged by the international standards
125043 community; therefore, this term, ``Epoch'', is used to abbreviate the reference to the actual
125044 standard, Coordinated Universal Time.

125045 **FIFO Special File**

125046 See Pipe (on page 3663).

125047 **File**

125048 It is permissible for an implementation-defined file type to be non-readable or non-writable.

125049 **File Classes**

125050 These classes correspond to the historical sets of permission bits. The classes are general to
125051 allow implementations flexibility in expanding the access mechanism for more stringent security
125052 environments. Note that a process is in one and only one class, so there is no ambiguity.

125053 **File Descriptor**

125054 Austin Group Defect 1493 is applied, moving some information from XCU Section 2.7 (on page
125055 2493) to this definition.

125056 **File Lock**

125057 Austin Group Defect 768 is applied, changing this definition.

125058 **File Mode**

125059 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0007 [834] is applied.

125060 **Filename**

125061 Filenames are sequences of bytes, not sequences of characters. The only bytes that this standard
125062 says cannot appear in any filename are the slash byte and the null byte. This is a side-effect of
125063 the fact that no conforming implementations of the standard currently provide a way to pass
125064 information specifying the locale associated with strings passed between user-level applications
125065 and the kernel. This decision could be revisited if implementations develop a way to associate a
125066 locale with the strings passed between kernel space and user space.

125067 Implementations may add other restrictions to the byte sequences allowed in filenames except
125068 that any filename consisting of no more than {NAME_MAX} bytes from the set of characters in
125069 the portable filename character set must be allowed.

125070 See Pathname (on page 3663).

125071 **File System**

125072 Historically, the meaning of this term has been overloaded with two meanings: that of the
125073 complete file hierarchy, and that of a mountable subset of that hierarchy; that is, a mounted file
125074 system. POSIX.1 uses the term ``file system'' in the second sense, except that it is limited to the
125075 scope of a process (and root directory of a process). This usage also clarifies the domain in which
125076 a file serial number is unique.

125077 **Foreground Job**

125078 Austin Group Defect 1254 is applied, changing this definition.

125079 **Graphic Character**

125080 This definition is made available for those definitions (in particular, *TZ*) which must exclude
125081 control characters.

125082 **Group Database**

125083 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/4 is applied, removing the words ``of
125084 implementation-defined format''. See User Database (on page 3675).

125085 **Group File***

125086 Implementation-defined; see User Database (on page 3675).

125087 **Group ID**

125088 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0008 [511] is applied.

125089 **Group Name**

125090 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0009 [584] is applied.

125091 **Hard Link**

125092 Austin Group Defect 1380 is applied, changing this definition.

125093 **Historical Implementations***

125094 This refers to previously existing implementations of programming interfaces and operating
125095 systems that are related to the interface specified by POSIX.1.

125096 **Hole**

125097 Austin Group Defect 415 is applied, adding this definition.

125098 **Hosted Implementation***

125099 This refers to a POSIX.1 implementation that is accomplished through interfaces from the
125100 POSIX.1 services to some alternate form of operating system kernel services. Note that the line
125101 between a hosted implementation and a native implementation is blurred, since most
125102 implementations will provide some services directly from the kernel and others through some
125103 indirect path. (For example, *fopen*( ) might use *open*( ); or *mkfifo*( ) might use *mknod*( ).) There is
125104 no necessary relationship between the type of implementation and its correctness, performance,
125105 and/or reliability.

125106 **Implementation***

125107 This term is generally used instead of its synonym, ``system'', to emphasize the consequences of
125108 decisions to be made by system implementors. Perhaps if no options or extensions to POSIX.1
125109 were allowed, this usage would not have occurred.

125110 The term ``specific implementation'' is sometimes used as a synonym for ``implementation''.
125111 This should not be interpreted too narrowly; both terms can represent a relatively broad group
125112 of systems. For example, a hardware vendor could market a very wide selection of systems that
125113 all used the same instruction set, with some systems desktop models and others large multi-user
125114 minicomputers. This wide range would probably share a common POSIX.1 operating system,
125115 allowing an application compiled for one to be used on any of the others; this is a [*specific*]
125116 *implementation*. However, such a wide range of machines probably has some differences
125117 between the models. Some may have different clock rates, different file systems, different
125118 resource limits, different network connections, and so on, depending on their sizes or intended
125119 usages. Even on two identical machines, the system administrators may configure them
125120 differently. Each of these different systems is known by the term ``a specific instance of a specific
125121 implementation''. This term is only used in the portions of POSIX.1 dealing with runtime
125122 queries: *sysconf*( ) and *pathconf*( ).

125123 **Incomplete Pathname***

125124 Absolute pathname has been adequately defined.

125125 **Interactive Device**

125126 Austin Group Defect 1347 is applied, adding a definition of interactive device.

Intrinsic Utility

Austin Group Defect 854 is applied, adding intrinsic utilities.

Job

Austin Group Defect 1254 is applied, changing this definition.

Job Control

In order to understand the job control facilities in POSIX.1 it is useful to understand how they are used by a job control-cognizant shell to create the user interface effect of job control.

While the job control facilities supplied by POSIX.1 can, in theory, support different types of interactive job control interfaces supplied by different types of shells, there was historically one particular interface that was most common when the standard was originally developed (provided by BSD C Shell).

This discussion describes that interface as a means of illustrating how the POSIX.1 job control facilities can be used.

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

The user can launch jobs (command pipelines) in either the foreground or background. When launched in the foreground, the shell waits for the job to complete before prompting for additional commands. When launched in the background, the shell does not wait, but immediately prompts for new commands.

If the user launches a job in the foreground and subsequently regrets this, the user can type the suspend character (typically set to <control>-Z), which causes the foreground process group to stop, and the shell to convert the corresponding foreground job to a suspended job and begin prompting for new commands. The suspended job can be continued by the user (via special shell commands) either as a foreground job or as a background job. Background jobs can also be moved into the foreground via shell commands.

If a background process group attempts to access the login terminal (controlling terminal), it is stopped by the terminal driver and the shell detects this and, in turn, suspends the corresponding background job and notifies the user. (Terminal access includes *read*( ) and certain terminal control functions, and conditionally includes *write*( ).) The user can continue the suspended job in the foreground, thus allowing the terminal access to succeed in an orderly fashion. After the terminal access succeeds, the user can optionally move the job into the background via the suspend character and shell commands.

Implementing Job Control Shells

The job control features of the POSIX shell (described in Section 2.11, on page 2518) and of other shells can be implemented using the job control facilities of the System Interfaces volume of POSIX.1-2024 in the following way.

The key feature necessary to provide job control is a way to group processes into jobs. This grouping is necessary in order to direct signals to a single job and also to identify which job is in the foreground. (There is at most one job that is in the foreground on any controlling terminal at a time.)

The concept of process groups is used to provide this grouping. The shell places the process(es) it creates for each job in a separate process group via the *setpgid*( ) function. To do this, the

setpgid( ) function is invoked by the shell for each process in the job. It is actually useful to invoke *setpgid*( ) twice for each process: once in the child process, after calling *fork*( ) to create the process, but before calling one of the *exec* family of functions to begin execution of the program, and once in the parent shell process, after calling *fork*( ) to create the child. The redundant invocation avoids a race condition by ensuring that the child process is placed into the new process group before either the parent or the child relies on this being the case. The process group ID for the job is selected by the shell to be equal to the process ID of one of the processes in the job. Some shells choose to make one process in the job be the parent of the other processes in the job (if any). Other shells (for example, the C Shell) choose to make themselves the parent of all processes in the job. In order to support this latter case, the *setpgid*( ) function accepts a process group ID parameter since the correct process group ID cannot be inherited from the shell.

The shell also controls which job is currently in the foreground. A foreground and background job differ in two ways: the shell waits for a foreground command to complete (or stop) before continuing to read new commands, and the terminal I/O driver inhibits terminal access by background jobs (causing the processes to stop). Thus, the shell must work cooperatively with the terminal I/O driver and have a common understanding of which job is currently in the foreground. It is the user who decides which command should be currently in the foreground, and the user informs the shell via shell commands. The shell, in turn, informs the terminal I/O driver via the *tcsetpgrp*( ) function. This indicates to the terminal I/O driver the process group ID of the foreground process group. When the current foreground job is either suspended or terminated, the shell places its own process group in the foreground via *tcsetpgrp*( ) before prompting for additional commands. Note that when a job is created the new process group begins as a background process group. It requires an explicit act of the shell via *tcsetpgrp*( ) to move a process group into the foreground.

When a process in a job stops or terminates, its parent (for example, the shell) receives synchronous notification by calling the *waitpid*( ) function with the WUNTRACED flag set. Asynchronous notification is also provided when the parent establishes a signal handler for SIGCHLD and does not specify the SA_NOCLDSTOP flag. Usually all processes in a job stop as a unit since the terminal I/O driver always sends job control stop signals to all processes in the process group.

To continue a suspended job, the shell sends a SIGCONT signal to the corresponding process group. In addition, if the job is being continued in the foreground, the shell invokes *tcsetpgrp*( ) to place the process group in the foreground before sending SIGCONT. Otherwise, the shell leaves itself in the foreground and reads additional commands.

There is additional flexibility in the POSIX.1 job control facilities that allows deviations from the typical interface. Clearing the TOSTOP terminal flag allows background jobs to perform *write*( ) functions without stopping. The same effect can be achieved on a per-process basis by having a process set the signal action for SIGTTOU to SIG_IGN.

A login session that is not using the job control facilities can be thought of as a large collection of processes that are all in the same job. Such a login session may have a partial distinction between foreground and background processes; that is, the shell waits for some processes before continuing to read new commands and does not wait for other processes. However, the terminal I/O driver considers all these processes to be in the foreground since they are all members of the same process group.

In addition to the basic job control operations already mentioned, a job control-cognizant shell needs to perform the following actions.

When a foreground (not background) job is suspended, the shell needs to sample and remember the current terminal settings so that it can restore them later when it continues the suspended

job in the foreground (via the *tcgetattr*() and *tcsetattr*() functions).

Because a shell itself can be spawned from a shell, it must take special action to ensure that child shells interact well with their parent shells. A child shell can be spawned to perform an interactive function (prompting the terminal for commands) or a non-interactive function (reading commands from a file). When operating non-interactively, the job control shell will by default refrain from performing the job control-specific actions described above. It will behave as a shell that does not support job control. For example, all jobs will be left in the same process group as the shell, which itself remains in the process group established for it by its parent. This allows the shell and its children to be treated as a single job by a parent shell, and they can be affected as a unit by terminal keyboard signals.

An interactive child shell can be spawned from another job control-cognizant shell in either the foreground or background. (For example, the user can execute an interactive shell in the background by means of the command `"sh &"`.) Before the child shell activates job control by calling *setpgid*() to place itself in its own process group and *tcsetpgrp*() to place its new process group in the foreground, it needs to ensure that it has already been placed in the foreground by its parent. (Otherwise, there could be multiple job control shells that simultaneously attempt to control mediation of the terminal.) To determine this, the shell retrieves its own process group via *getpgrp*() and the process group of the current foreground job via *tcgetpgrp*(). If these are not equal, the shell sends SIGTTIN to its own process group, causing itself to stop. When continued later by its parent, the shell repeats the process group check. When the process groups finally match, the shell is in the foreground and it can proceed to take control. After this point, the shell ignores all the job control stop signals so that it does not inadvertently stop itself.

Implementing Job Control Applications

Most applications do not need to be aware of job control signals and operations; the intuitively correct behavior happens by default. However, sometimes an application can inadvertently interfere with normal job control processing, or an application may choose to overtly effect job control in cooperation with normal shell procedures.

An application can inadvertently subvert job control processing by ``blindly'' altering the handling of signals. A common application error is to learn how many signals the system supports and to ignore or catch them all. Such an application makes the assumption that it does not know what this signal is, but knows the right handling action for it. The system may initialize the handling of job control stop signals so that they are being ignored. This allows shells that do not support job control to inherit and propagate these settings and hence to be immune to stop signals. A job control shell will set the handling to the default action and propagate this, allowing processes to stop. In doing so, the job control shell is taking responsibility for restarting the stopped applications. If an application wishes to catch the stop signals itself, it should first determine their inherited handling states. If a stop signal is being ignored, the application should continue to ignore it. This is directly analogous to the recommended handling of SIGINT described in the referenced UNIX Programmer's Manual.

If an application is reading the terminal and has disabled the interpretation of special characters (by clearing the ISIG flag), the terminal I/O driver will not send SIGTSTP when the suspend character is typed. Such an application can simulate the effect of the suspend character by recognizing it and sending SIGTSTP to its process group as the terminal driver would have done. Note that the signal is sent to the process group, not just to the application itself; this ensures that other processes in the job also stop. (Note also that other processes in the job could be children, siblings, or even ancestors.) Applications should not assume that the suspend character is <control>-Z (or any particular value); they should retrieve the current setting at startup.

125268 *Implementing Job Control Systems*

125269 The intent in adding 4.2 BSD-style job control functionality was to adopt the necessary 4.2 BSD
125270 programmatic interface with only minimal changes to resolve syntactic or semantic conflicts
125271 with System V or to close recognized security holes. The goal was to maximize the ease of
125272 providing both conforming implementations and Conforming POSIX.1 Applications.

125273 It is only useful for a process to be affected by job control signals if it is the descendant of a job
125274 control shell. Otherwise, there will be nothing that continues the stopped process.

125275 POSIX.1 does not specify how controlling terminal access is affected by a user logging out (that
125276 is, by a controlling process terminating). 4.2 BSD uses the *vhangup*( ) function to prevent any
125277 access to the controlling terminal through file descriptors opened prior to logout. System V does
125278 not prevent controlling terminal access through file descriptors opened prior to logout (except
125279 for the case of the special file, **/dev/tty**). Some implementations choose to make processes
125280 immune from job control after logout (that is, such processes are always treated as if in the
125281 foreground); other implementations continue to enforce foreground/background checks after
125282 logout. Therefore, a Conforming POSIX.1 Application should not attempt to access the
125283 controlling terminal after logout since such access is unreliable. If an implementation chooses to
125284 deny access to a controlling terminal after its controlling process exits, POSIX.1 requires a certain
125285 type of behavior (see Controlling Terminal, on page 3651).

125286 Austin Group Defect 1254 is applied, changing this definition.

125287 **Job ID**

125288 Austin Group Defect 1254 is applied, changing ``job control job ID'' to ``job ID''.

125289 **Joinable Thread**

125290 Austin Group Defect 792 is applied, adding this definition.

125291 **Kernel***

125292 See System Call* (on page 3673).

125293 **Library Routine***

125294 See System Call* (on page 3673).

125295 **Link**

125296 Austin Group Defect 1380 is applied, changing this definition.

125297 **Live Process**

125298 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0010 [690] is applied.

125299 **Live Thread**

125300 Austin Group Defect 792 is applied, adding this definition.

125301 **Logical Device***

125302 Implementation-defined.

125303 **Map**

125304 The definition of map is included to clarify the usage of mapped pages in the description of the
125305 behavior of process memory locking.

125306 **Memory-Resident**

125307 The term ``memory-resident'' is historically understood to mean that the so-called resident pages
125308 are actually present in the physical memory of the computer system and are immune from
125309 swapping, paging, copy-on-write faults, and so on. This is the actual intent of POSIX.1-2024 in
125310 the process memory locking section for implementations where this is logical. But for some
125311 implementations—primarily mainframes—actually locking pages into primary storage is not
125312 advantageous to other system objectives, such as maximizing throughput. For such
125313 implementations, memory locking is a ``hint'' to the implementation that the application wishes
125314 to avoid situations that would cause long latencies in accessing memory. Furthermore, there are
125315 other implementation-defined issues with minimizing memory access latencies that ``memory
125316 residency'' does not address—such as MMU reload faults. The definition attempts to
125317 accommodate various implementations while allowing conforming applications to specify to the
125318 implementation that they want or need the best memory access times that the implementation
125319 can provide.

125320 **Memory Object***

125321 The term ``memory object'' usually implies shared memory. If the object is the same as a
125322 filename in the file system name space of the implementation, it is expected that the data written
125323 into the memory object be preserved on disk. A memory object may also apply to a physical
125324 device on an implementation. In this case, writes to the memory object are sent to the controller
125325 for the device and reads result in control registers being returned.

125326 **Messages Object**

125327 Austin Group Defect 1122 is applied, adding this definition.

125328 **Mounted File System***

125329 See File System (on page 3654).

125330 **Multi-Threaded Library**

125331 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0011 [625] is applied.

125332 **Multi-Threaded Process**

125333 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0011 [625] is applied.

125334 **Multi-Threaded Program**

125335 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0011 [625] is applied.

125336 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

125337 **Mutex**

125338 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

125339 **Name**

125340 There are no explicit limits in POSIX.1-2024 on the sizes of names, words (see the definition of
125341 word in the Base Definitions volume of POSIX.1-2024), lines, or other objects. However, other
125342 implicit limits do apply: shell script lines produced by many of the standard utilities cannot
125343 exceed {LINE_MAX} and the sum of exported variables comes under the {ARG_MAX} limit.
125344 Historical shells dynamically allocate memory for names and words and parse incoming lines a
125345 character at a time. Lines cannot have an arbitrary {LINE_MAX} limit because of historical
125346 practice, such as makefiles, where *make* removes the <newline> characters associated with the
125347 commands for a target and presents the shell with one very long line. The text on INPUT FILES
125348 in XCU Section 1.4 (on page 2462) does allow a shell to run out of memory, but it cannot have
125349 arbitrary programming limits.

125350 **Native Implementation***

125351 This refers to an implementation of POSIX.1 that interfaces directly to an operating system
125352 kernel; see also *hosted implementation*. A similar concept is a native UNIX system, which would
125353 be a kernel derived from one of the original UNIX system products.

125354 **Negative**

125355 Austin Group Defect 1428 is applied, adding this definition.

125356 **Nice Value**

125357 This definition is not intended to suggest that all processes in a system have priorities that are
125358 comparable. Scheduling policy extensions, such as adding realtime priorities, make the notion of
125359 a single underlying priority for all scheduling policies problematic. Some implementations may
125360 implement the features related to *nice* to affect all processes on the system, others to affect just
125361 the general time-sharing activities implied by POSIX.1-2024, and others may have no effect at all.
125362 Because of the use of ``implementation-defined'' in *nice* and *renice*, a wide range of
125363 implementation strategies is possible.

125364 **Null Pointer**

125365 Austin Group Defect 940 is applied, adding a statement that any pointer object whose
125366 representation has all bits set to zero will be interpreted as a null pointer.

125367 **Null Terminator**

125368 Austin Group Defect 1621 is applied, adding this definition.

125369 **OFD-Owned File Lock**

125370 Austin Group Defect 768 is applied, adding this definition.

125371 **Open File Description**

125372 An ``open file description'', as it is currently named, describes how a file is being accessed. What
125373 is currently called a ``file descriptor'' is actually just an identifier or ``handle''; it does not
125374 actually describe anything.

125375 The following alternate names were discussed:

125376 • For ``open file description'':
125377 ``open instance'', ``file access description'', ``open file information'', and ``file access
125378 information''.

125379 • For ``file descriptor'':
125380 ``file handle'', ``file number'' (cf., *fileno*()). Some historical implementations use the term
125381 ``file table entry''.

125382 **Option-Argument**

125383 Austin Group Defect 1784 is applied, changing this definition.

125384 **Orphaned Process Group**

125385 Historical implementations have a concept of an orphaned process, which is a process whose
125386 parent process has exited. When job control is in use, it is necessary to prevent processes from
125387 being stopped in response to interactions with the terminal after they no longer are controlled by
125388 a job control-cognizant program. Because signals generated by the terminal are sent to a process
125389 group and not to individual processes, and because a signal may be provoked by a process that
125390 is not orphaned, but sent to another process that is orphaned, it is necessary to define an
125391 orphaned process group. The definition assumes that a process group will be manipulated as a
125392 group and that the job control-cognizant process controlling the group is outside of the group
125393 and is the parent of at least one process in the group (so that state changes may be reported via
125394 *waitpid*()). Therefore, a group is considered to be controlled as long as at least one process in the
125395 group has a parent that is outside of the process group, but within the session.

125396 This definition of orphaned process groups ensures that a session leader's process group is
125397 always considered to be orphaned, and thus it is prevented from stopping in response to
125398 terminal signals.

125399 **Page**

125400 The term ``page'' is defined to support the description of the behavior of memory mapping for
125401 shared memory and memory mapped files, and the description of the behavior of process
125402 memory locking. It is not intended to imply that shared memory/file mapping and memory
125403 locking are applicable only to ``paged'' architectures. For the purposes of POSIX.1-2024,
125404 whatever the granularity on which an architecture supports mapping or locking, this is
125405 considered to be a ``page'' . If an architecture cannot support the memory mapping or locking
125406 functions specified by POSIX.1-2024 on any granularity, then these options will not be
125407 implemented on the architecture.

125408 **Pathname**

125409 Pathnames historically allowed all bytes except for the <slash> and <NUL> characters. For
125410 compatibility with existing file systems, this usage is maintained throughout the standard by
125411 noting that a pathname need not be a valid character string in all locales. However, the
125412 properties of the portable filename character set are such that a pathname using only those
125413 characters and the <slash> is portable in all locales as a character string.

125414 Austin Group Defect 1073 is applied, making it implementation-defined whether the case of
125415 exactly two leading <slash> characters is treated specially.

125416 **Passwd File***

125417 Implementation-defined; see User Database (on page 3675).

125418 **Parent Directory**

125419 There may be more than one directory entry pointing to a given directory in some
125420 implementations. The wording here identifies that exactly one of those is the parent directory. In
125421 pathname resolution, dot-dot is identified as the way that the unique directory is identified.
125422 (That is, the parent directory is the one to which dot-dot points.) In the case of a remote file
125423 system, if the same file system is mounted several times, it would appear as if they were distinct
125424 file systems (with interesting synchronization properties).

125425 **Pattern**

125426 Austin Group Defect 1443 is applied, changing this definition to be inclusive of all uses of shell
125427 pattern matching notation.

125428 **Pipe**

125429 It proved convenient to define a pipe as a special case of a FIFO, even though historically the
125430 latter was not introduced until System III and does not exist at all in 4.3 BSD.

125431 **Portable Filename**

125432 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0012 [584] is applied.

125433 **Portable Filename Character Set**

125434 The encoding of this character set is not specified—specifically, ASCII is not required. But the
125435 implementation must provide a unique character code for each of the printable graphics
125436 specified by POSIX.1; see also Section A.4.9 (on page 3678).

125437 Situations where characters beyond the portable filename character set (or historically ASCII or
125438 the ISO/IEC 646:1991 standard) would be used (in a context where the portable filename
125439 character set or the ISO/IEC 646:1991 standard is required by POSIX.1) are expected to be
125440 common. Although such a situation renders the use technically non-compliant, mutual
125441 agreement among the users of an extended character set will make such use portable between
125442 those users. Such a mutual agreement could be formalized as an optional extension to POSIX.1.
125443 (Making it required would eliminate too many possible systems, as even those systems using the
125444 ISO/IEC 646:1991 standard as a base character set extend their character sets for Western
125445 Europe and the rest of the world in different ways.)

125446 Nothing in POSIX.1 is intended to preclude the use of extended characters where interchange is
125447 not required or where mutual agreement is obtained. It has been suggested that in several places
125448 ``should'' be used instead of ``shall''. Because (in the worst case) use of any character beyond the
125449 portable filename character set would render the program or data not portable to all possible

125450 systems, no extensions are permitted in this context.

125451 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0013 [584] is applied.

125452 **Portable Messages Object Source File (or Dot-Po File)**

125453 Austin Group Defect 1122 is applied, adding this definition.

125454 **Positional Parameter**

125455 Austin Group Defect 1514 is applied, changing this definition in line with earlier changes to the
125456 cross-reference to which it refers.

125457 **Positive**

125458 Austin Group Defect 1428 is applied, adding this definition.

125459 **Process**

125460 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0014 [690] is applied.

125461 **Process Lifetime**

125462 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/5 is applied, adding *fork*( ), *posix_spawn*( ),
125463 *posix_spawnp*( ), and *vfork*( ) to the list of functions.

125464 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0014 [690] is applied.

125465 **Process Termination**

125466 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/6 is applied, rewording the definition to
125467 address the ``passive exit'' on termination of the last thread or the *_Exit*( ) function.

125468 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0014 [690] is applied.

125469 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899: 2018 standard.

125470 **Process-Owned File Lock**

125471 Austin Group Defect 768 is applied, adding this definition.

125472 **Pseudo-Terminal**

125473 Austin Group Defect 1466 is applied, changing the terminology used for pseudo-terminal
125474 devices.

125475 **Radix Character (or Decimal-Point Character)**

125476 Austin Group Defect 1449 is applied, adding ``(or Decimal-Point Character)''.

125477 **Record Lock**

125478 Austin Group Defect 768 is applied, adding this definition.

125479 **Regular Built-In Utility (or Regular Built-In)**

125480 Austin Group Defect 850 is applied, adding this entry as a pointer to the *Built-In Utility*
125481 definition.

125482 **Regular File**

125483 POSIX.1 does not intend to preclude the addition of structuring data (for example, record
125484 lengths) in the file, as long as such data is not visible to an application that uses the features
125485 described in POSIX.1.

125486 **Root Directory**

125487 This definition permits the operation of *chroot*( ), even though that function is not in POSIX.1; see
125488 also Section A.4.8 (on page 3678).

125489 **Root File System***

125490 Implementation-defined.

125491 **Root of a File System***

125492 Commonly used to refer to a mount point; this standard uses the latter.

125493 **Signal**

125494 The definition implies a double meaning for the term. Although a signal is an event, common
125495 usage implies that a signal is an identifier of the class of event.

125496 **Single-Threaded Process**

125497 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0011 [625] is applied.

125498 **Single-Threaded Program**

125499 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0011 [625] is applied.

125500 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

125501 **Source Code**

125502 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0015 [896] is applied.

125503 **Sparse File**

125504 Austin Group Defect 415 is applied, adding this definition.

125505 **Special Built-In Utility (or Special Built-In)**

125506 Austin Group Defect 1583 is applied, clarifying that ``special built-in utility'' and ``special built-
125507 in'' are equivalent terms.

125508 **Standard Error**

125509 Austin Group Defect 1493 is applied, expanding this definition to cover uses of the term outside
125510 the XSH volume.

125511 **Standard Input**

125512 Austin Group Defect 1493 is applied, expanding this definition to cover uses of the term outside
125513 the XSH volume.

125514 **Standard Output**

125515 Austin Group Defect 1493 is applied, expanding this definition to cover uses of the term outside
125516 the XSH volume.

125517 **Stream**

125518 Austin Group Defect 1371 is applied, updating the stream definition so that it applies to the shell
125519 command language as well as the C language.

125520 **Superuser***

125521 This concept, with great historical significance to UNIX system users, has been replaced with the
125522 notion of appropriate privileges.

125523 **Supplementary Group ID**

125524 The POSIX.1-1990 standard is inconsistent in its treatment of supplementary groups. The
125525 definition of supplementary group ID explicitly permits the effective group ID to be included in
125526 the set, but wording in the description of the *setuid*() and *setgid*() functions states: ``Any
125527 supplementary group IDs of the calling process remain unchanged by these function calls''. In
125528 the case of *setgid*() this contradicts that definition. In addition, some felt that the unspecified
125529 behavior in the definition of supplementary group IDs adds unnecessary portability problems.
125530 The standard developers considered several solutions to this problem:

125531 1. Reword the description of *setgid*() to permit it to change the supplementary group IDs to
125532 reflect the new effective group ID. A problem with this is that it adds more ``may''s to the
125533 wording and does not address the portability problems of this optional behavior.

125534 2. Mandate the inclusion of the effective group ID in the supplementary set (giving
125535 {NGROUPS_MAX} a minimum value of 1). This is the behavior of 4.4 BSD. In that
125536 system, the effective group ID is the first element of the array of supplementary group
125537 IDs (there is no separate copy stored, and changes to the effective group ID are made only
125538 in the supplementary group set). By convention, the initial value of the effective group ID
125539 is duplicated elsewhere in the array so that the initial value is not lost when executing a
125540 set-group-ID program.

125541 3. Change the definition of supplementary group ID to exclude the effective group ID and
125542 specify that the effective group ID does not change the set of supplementary group IDs.
125543 This is the behavior of 4.2 BSD, 4.3 BSD, and System V Release 4.

125544 4. Change the definition of supplementary group ID to exclude the effective group ID, and
125545 require that *getgroups*() return the union of the effective group ID and the supplementary
125546 group IDs.

125547 5. Change the definition of {NGROUPS_MAX} to be one more than the number of
125548 supplementary group IDs, so it continues to be the number of values returned by
125549 *getgroups*() and existing applications continue to work. This alternative is effectively the

125550 same as the second (and might actually have the same implementation).

125551 The standard developers decided to permit either 2 or 3. The effective group ID is orthogonal to
125552 the set of supplementary group IDs, and it is implementation-defined whether *getgroups*()
125553 returns this. If the effective group ID is returned with the set of supplementary group IDs, then
125554 all changes to the effective group ID affect the supplementary group set returned by *getgroups*().
125555 It is permissible to eliminate duplicates from the list returned by *getgroups*(). However, if a
125556 group ID is contained in the set of supplementary group IDs, setting the group ID to that value
125557 and then to a different value should not remove that value from the supplementary group IDs.

125558 The definition of supplementary group IDs has been changed to not include the effective group
125559 ID. This simplifies permanent rationale and makes the relevant functions easier to understand.
125560 The *getgroups*() function has been modified so that it can, on an implementation-defined basis,
125561 return the effective group ID. By making this change, functions that modify the effective group
125562 ID do not need to discuss adding to the supplementary group list; the only view into the
125563 supplementary group list that the application developer has is through the *getgroups*() function.

125564 **Suspended Job**

125565 Austin Group Defect 1254 is applied, changing this definition.

125566 **Symbolic Constant**

125567 Earlier versions of this standard used a variety of terms other than ``macro'' for many of the
125568 constants defined in headers, and it was not clear in which of these cases they were required to
125569 be macros or not, or to be pre-processor constants (i.e., usable in **#if**) or not. In cases where the
125570 symbols had a reserved prefix or suffix, there was often inconsistency between whether the
125571 prefix/suffix was reserved only for macros or for any use, and whether the term ``macro'' or a
125572 different term was used in the descriptions of the symbols. There were also some unintentional
125573 differences from the ISO C standard.

125574 One of the most commonly used terms was ``symbolic constant''. This has now been designated
125575 as the default term to be used wherever appropriate, and a formal definition of the term has
125576 been added giving the exact requirements for symbols that are described as symbolic constants.

125577 The standard developers have performed a major rationalization of the header descriptions of
125578 symbols with constant values according to the following policy:

125579 • Where symbols are from the ISO C standard, the wording from the ISO C standard (or
125580 equivalent, in cases where the exact wording is not appropriate) is used to describe them.

125581 • For all other constants, the first choice is to use ``symbolic constant'' when the
125582 requirements for the symbol are a reasonably close fit with those of the term.

125583 The description of the symbol can override individual requirements for symbolic
125584 constants; e.g., to specify a non-integer type, or to add a requirement that the symbol is
125585 usable in **#if** preprocessor directives.

125586 • When neither of the above apply, the exact requirements are stated in the description.
125587 (Note that macros are not required to be usable in **#if**, or even to expand to constant
125588 expressions, unless explicitly stated.)

125589 • In cases where there is a reserved prefix or suffix, if the symbol(s) with that prefix/suffix
125590 are from the ISO C standard and are required to be macros, or if the symbol is required to
125591 be usable in **#if**, then the prefix/suffix is reserved for use only as macros. If the symbol(s)
125592 are ``symbolic constants'' and not required to be usable in **#if**, the prefix/suffix is reserved
125593 for any use except in a few special cases.

125594 Where a constant is required to be a macro but is also allowed to be another type of constant

such as an enumeration constant, on implementations which do define it as another type of constant the macro is typically defined as follows:

```
#define macro_name macro_name
```

This allows applications to use **#ifdef**, etc. to determine whether the macro is defined, but the macro is not usable in **#if** preprocessor directives because the preprocessor will treat the unexpanded word *macro_name* as having the value zero.

Symbolic Link

Earlier versions of this standard did not require symbolic links to have attributes such as ownership and a file serial number. This was because the 4.4 BSD implementation did not have them, and it was expected that other implementations may wish to do the same. However, experience with 4.4 BSD has shown that symbolic links implemented in this way cause problems for users and application developers, and later BSD systems have reverted to using *inode*s to implement symbolic links. Allowing no-*inode* symbolic links also caused problems in the standard. For example, leaving the *st_ino* value for symbolic links unspecified meant that the common technique of comparing the *st_dev* and *st_ino* values for two pathnames to see if they refer to the same file could only be used with *stat*() in conforming applications and not with *lstat*(). The standard now requires symbolic links to have meaningful values for the same **struct stat** fields as regular files, except for the file mode bits in *st_mode*. Historically, the file mode bits were unused (the contents of a symbolic link could always be read), but implementations differed as to whether the file mode bits (as returned in *st_mode* or reported by *ls* **–l**) were set according to the *umask* or just to a fixed value such as 0777. Accordingly, the standard requires the file mode bits to be ignored by *readlink*() and when a symbolic link is followed during pathname resolution, but leaves the corresponding part of the value returned in *st_mode* unspecified.

Historical implementations were followed when determining which interfaces should apply to symbolic links. Interfaces that historically followed symbolic links include *chmod*(), *stat*(), and *utime*(). Interfaces that historically did not follow symbolic links include *lstat*(), *rename*(), *remove*(), *rmdir*(), and *unlink*(). For *chown*() and *link*(), historical implementations differed. POSIX.1-2024 inherited the *lchown*() function from the Single UNIX Specification, Version 2, and therefore requires *chown*() to follow symbolic links. Earlier versions of this standard required *link*() to follow symbolic links, but with the addition of the *linkat*() function (which has a flag to indicate whether to follow symbolic links), both behaviors are now allowed for *link*().

When the final component of a pathname is a symbolic link, the standard requires that a trailing <slash> causes the link to be followed. This is the behavior of historical implementations. For example, for **/a/b** and **/a/b/**, if **/a/b** is a symbolic link to a directory, then **/a/b** refers to the symbolic link, and **/a/b/** refers to the directory to which the symbolic link points.

Because a symbolic link and its referenced object coexist in the file system name space, confusion can arise in distinguishing between the link itself and the referenced object. Historically, utilities and system calls have adopted their own link following conventions in a somewhat *ad hoc* fashion. Rules for a uniform approach are outlined here, although historical practice has been adhered to as much as was possible. To promote consistent system use, user-written utilities are encouraged to follow these same rules.

Symbolic links are handled either by operating on the link itself, or by operating on the object referenced by the link. In the latter case, an application or system call is said to ``follow'' the link. Symbolic links may reference other symbolic links, in which case links are dereferenced until an object that is not a symbolic link is found, a symbolic link that references a file that does not exist is found, or a loop is detected. (Current implementations do not detect loops, but have a limit on the number of symbolic links that they will dereference before declaring it an error.)

There are four domains for which default symbolic link policy is established in a system. In almost all cases, there are utility options that override this default behavior. The four domains are as follows:

1. Symbolic links specified to system calls that take pathname arguments
2. Symbolic links specified as command line pathname arguments to utilities that are not performing a traversal of a file hierarchy
3. Symbolic links referencing files not of type directory, specified to utilities that are performing a traversal of a file hierarchy
4. Symbolic links referencing files of type directory, specified to utilities that are performing a traversal of a file hierarchy

First Domain

The first domain is considered in earlier rationale.

Second Domain

The reason this category is restricted to utilities that are not traversing the file hierarchy is that some standard utilities take an option that specifies a hierarchical traversal, but by default operate on the arguments themselves. Generally, users specifying the option for a file hierarchy traversal wish to operate on a single, physical hierarchy, and therefore symbolic links, which may reference files outside of the hierarchy, are ignored. For example, *chown owner file* is a different operation from the same command with the **−R** option specified. In this example, the behavior of the command *chown owner file* is described here, while the behavior of the command *chown* **−R** *owner file* is described in the third and fourth domains.

The general rule is that the utilities in this category follow symbolic links named as arguments.

Exceptions in the second domain are:

- The *mv* and *rm* utilities do not follow symbolic links named as arguments, but respectively attempt to rename or delete them.
- The *ls* utility is also an exception to this rule. For compatibility with historical systems, when the **−R** option is not specified, the *ls* utility follows symbolic links named as arguments if the **−L** option is specified or if the **−F**, **−d**, or **−l** options are not specified. (If the **−L** option is specified, *ls* always follows symbolic links; it is the only utility where the **−L** option affects its behavior even though a tree walk is not being performed.)

All other standard utilities, when not traversing a file hierarchy, always follow symbolic links named as arguments.

Historical practice is that the **−h** option is specified if standard utilities are to act upon symbolic links instead of upon their targets. Examples of commands that have historically had a **−h** option for this purpose are the *chgrp*, *chown*, *file*, and *test* utilities.

Third Domain

The third domain is symbolic links, referencing files not of type directory, specified to utilities that are performing a traversal of a file hierarchy. (This includes symbolic links specified as command line pathname arguments or encountered during the traversal.)

The intention of the Shell and Utilities volume of POSIX.1-2024 is that the operation that the utility is performing is applied to the symbolic link itself, if that operation is applicable to symbolic links. If the operation is not applicable to symbolic links, the symbolic link should be ignored. Specifically, by default, no change should be made to the file referenced by the symbolic link.

Fourth Domain

The fourth domain is symbolic links referencing files of type directory, specified to utilities that are performing a traversal of a file hierarchy. (This includes symbolic links specified as command line pathname arguments or encountered during the traversal.)

Most standard utilities do not, by default, indirect into the file hierarchy referenced by the symbolic link. (The Shell and Utilities volume of POSIX.1-2024 uses the informal term ``physical walk'' to describe this case. The case where the utility does indirect through the symbolic link is termed a ``logical walk''.)

There are three reasons for the default to be a physical walk:

1. With very few exceptions, a physical walk has been the historical default on UNIX systems supporting symbolic links. Because some utilities (that is, *rm*) must default to a physical walk, regardless, changing historical practice in this regard would be confusing to users and needlessly incompatible.

2. For systems where symbolic links have the historical file attributes (that is, *owner*, *group*, *mode*), defaulting to a logical traversal would require the addition of a new option to the commands to modify the attributes of the link itself. This is painful and more complex than the alternatives.

3. There is a security issue with defaulting to a logical walk. Historically, the command *chown* **–R** *user file* has been safe for the superuser because *setuid* and *setgid* bits were lost when the ownership of the file was changed. If the walk were logical, changing ownership would no longer be safe because a user might have inserted a symbolic link pointing to any file in the tree. Again, this would necessitate the addition of an option to the commands doing hierarchy traversal to not indirect through the symbolic links, and historical scripts doing recursive walks would instantly become security problems. While this is mostly an issue for system administrators, it is preferable to not have different defaults for different classes of users.

However, the standard developers agreed to leave it unspecified to achieve consensus.

As consistently as possible, users may cause standard utilities performing a file hierarchy traversal to follow any symbolic links named on the command line, regardless of the type of file they reference, by specifying the **–H** (for half logical) option. This option is intended to make the command line name space look like the logical name space.

As consistently as possible, users may cause standard utilities performing a file hierarchy traversal to follow any symbolic links named on the command line as well as any symbolic links encountered during the traversal, regardless of the type of file they reference, by specifying the **–L** (for logical) option. This option is intended to make the entire name space look like the logical name space.

For consistency, implementors are encouraged to use the **–P** (for ``physical'') flag to specify the physical walk in utilities that do logical walks by default for whatever reason.

When one or more of the **–H**, **–L**, and **–P** flags can be specified, the last one specified determines the behavior of the utility. This permits users to alias commands so that the default behavior is a logical walk and then override that behavior on the command line.

Exceptions in the Third and Fourth Domains

The *ls* and *rm* utilities are exceptions to these rules. The *rm* utility never follows symbolic links and does not support the **–H**, **–L**, or **–P** options. Some historical versions of *ls* always followed symbolic links given on the command line whether the **–L** option was specified or not. Historical versions of *ls* did not support the **–H** option. In POSIX.1-2024, unless one of the **–H** or

125733 –**L** options is specified, the *ls* utility only follows symbolic links to directories that are given as
125734 operands. The *ls* utility does not support the –**P** option.

125735 The Shell and Utilities volume of POSIX.1-2024 requires that the standard utilities *ls*, *find*, and
125736 *pax* detect infinite loops when doing logical walks; that is, a directory, or more commonly a
125737 symbolic link, that refers to an ancestor in the current file hierarchy. If the file system itself is
125738 corrupted, causing the infinite loop, it may be impossible to recover. Because *find* and *ls* are often
125739 used in system administration and security applications, they should attempt to recover and
125740 continue as best as they can. The *pax* utility should terminate because the archive it was creating
125741 is by definition corrupted. Other, less vital, utilities should probably simply terminate as well.
125742 Implementations are strongly encouraged to detect infinite loops in all utilities.

125743 Historical practice is shown in Table A-1 (on page 3672). The heading **SVID3** stands for the
125744 Third Edition of the System V Interface Definition.

125745 Historically, several shells have had built-in versions of the *pwd* utility. In some of these shells,
125746 *pwd* reported the physical path, and in others, the logical path. Implementations of the shell
125747 corresponding to POSIX.1-2024 must report the logical path by default.

125748 The *cd* command is required, by default, to treat the filename dot-dot logically. Implementors are
125749 required to support the –**P** flag in *cd* so that users can have their current environment handled
125750 physically. In 4.3 BSD, *chgrp* during tree traversal changed the group of the symbolic link, not
125751 the target. Symbolic links in 4.4 BSD did not have *owner*, *group*, *mode*, or other standard UNIX
125752 system file attributes.

125753 **Table A-1** Historical Practice for Symbolic Links

125754

Utility	SVID3	4.3 BSD	4.4 BSD	POSIX	Comments
125755 *cd*				**–L**	Treat "**..**" logically.
125756 *cd*				**–P**	Treat "**..**" physically.
125757 *chgrp*			**–H**	**–H**	Follow command line symlinks.
125758 *chgrp*			**–h**	**–L**	Follow symlinks.
125759 *chgrp*	**–h**			**–h**	Affect the symlink.
125760 *chmod*					Affect the symlink.
125761 *chmod*			**–H**		Follow command line symlinks.
125762 *chmod*			**–h**		Follow symlinks.
125763 *chown*			**–H**	**–H**	Follow command line symlinks.
125764 *chown*			**–h**	**–L**	Follow symlinks.
125765 *chown*	**–h**			**–h**	Affect the symlink.
125766 *cp*			**–H**	**–H**	Follow command line symlinks.
125767 *cp*			**–h**	**–L**	Follow symlinks.
125768 *cpio*	**–L**		**–L**		Follow symlinks.
125769 *du*			**–H**	**–H**	Follow command line symlinks.
125770 *du*			**–h**	**–L**	Follow symlinks.
125771 *file*	**–h**			**–h**	Affect the symlink.
125772 *find*			**–H**	**–H**	Follow command line symlinks.
125773 *find*			**–h**	**–L**	Follow symlinks.
125774 *find*	**–follow**		**–follow**		Follow symlinks.
125775 *ln*	**–s**	**–s**	**–s**	**–s**	Create a symbolic link.
125776 *ls*	**–L**	**–L**	**–L**	**–L**	Follow symlinks.
125777 *ls*				**–H**	Follow command line symlinks.
125778 *mv*					Operates on the symlink.
125779 *pax*			**–H**	**–H**	Follow command line symlinks.
125780 *pax*			**–h**	**–L**	Follow symlinks.
125781 *pwd*				**–L**	Printed path may contain symlinks.
125782 *pwd*				**–P**	Printed path will not contain symlinks.
125783 *rm*					Operates on the symlink.
125784 *tar*			**–H**		Follow command line symlinks.
125785 *tar*		**–h**	**–h**		Follow symlinks.
125786 *test*	**–h**		**–h**	**–h**	Affect the symlink.

125787 **Synchronized I/O Data Integrity Completion**

125788 Austin Group Defect 672 is applied, clarifying how this definition applies to directories, and that
125789 it does not apply to symbolic links.

125790 **Synchronously-Generated Signal**

125791 Those signals that may be generated synchronously include SIGABRT, SIGBUS, SIGILL, SIGFPE,
125792 SIGPIPE, and SIGSEGV.

125793 Any signal sent via the *raise*() function or a *kill*() function targeting the current process is also
125794 considered synchronous.

125795 **System Call***

125796 The distinction between a ``system call'' and a ``library routine'' is an implementation detail that
125797 may differ between implementations and has thus been excluded from POSIX.1.

125798 See ``Interface, Not Implementation'' in the Preface.

125799 **System Console**

125800 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/7 is applied, changing from ``An
125801 implementation-defined device'' to ``A device''.

125802 **System Databases**

125803 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/9 is applied, rewording the definition to
125804 reference the existing definitions for ``group database'' and ``user database''.

125805 **System Process**

125806 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/8 is applied, rewording the definition to
125807 remove the requirement for an implementation to define the object.

125808 **System Reboot**

125809 A ``system reboot'' is an event initiated by an unspecified circumstance that causes all processes
125810 (other than special system processes) to be terminated in an implementation-defined manner,
125811 after which any changes to the state and contents of files created or written to by a Conforming
125812 POSIX.1 Application prior to the event are implementation-defined.

125813 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/10 is applied, changing ``An
125814 implementation-defined sequence of events'' to ``An unspecified sequence of events''.

125815 **Synchronized I/O Data (and File) Integrity Completion**

125816 These terms specify that for synchronized read operations, pending writes must be successfully
125817 completed before the read operation can complete. This is motivated by two circumstances.
125818 Firstly, when synchronizing processes can access the same file, but not share common buffers
125819 (such as for a remote file system), this requirement permits the reading process to guarantee that
125820 it can read data written remotely. Secondly, having data written synchronously is insufficient to
125821 guarantee the order with respect to a subsequent write by a reading process, and thus this extra
125822 read semantic is necessary.

125823 **Text Domain**

125824 Austin Group Defect 1122 is applied, adding this definition.

125825 **Text File**

125826 The term ``text file'' does not prevent the inclusion of control or other non-printable characters
125827 (other than NUL). Therefore, standard utilities that list text files as inputs or outputs are either
125828 able to process the special characters or they explicitly describe their limitations within their
125829 individual descriptions. The definition of ``text file'' has caused controversy. The only difference
125830 between text and binary files is that text files have lines of less than {LINE_MAX} bytes, with no
125831 NUL characters, each terminated by a <newline>. The definition allows a file with a single
125832 <newline>, or a totally empty file, to be called a text file. If a file ends with an incomplete line it
125833 is not strictly a text file by this definition. The <newline> referred to in POSIX.1-2024 is not some
125834 generic line separator, but a single character; files created on systems where they use multiple
125835 characters for ends of lines are not portable to all conforming systems without some translation

125836 process unspecified by POSIX.1-2024.

125837 **Thread**

125838 POSIX.1-2024 defines a live thread to be a flow of control within a process. Each thread has a
125839 minimal amount of private state; most of the state associated with a process is shared among all
125840 of the threads in the process. While most multi-thread extensions to POSIX have taken this
125841 approach, others have made different decisions.

125842 **Note:** The choice to put threads within a process does not constrain implementations to implement
125843 threads in that manner. However, all functions have to behave as though threads share the
125844 indicated state information with the process from which they were created.

125845 Threads need to share resources in order to cooperate. Memory has to be widely shared between
125846 threads in order for the threads to cooperate at a fine level of granularity. Threads keep data
125847 structures and the locks protecting those data structures in shared memory. For a data structure
125848 to be usefully shared between threads, such structures should not refer to any data that can only
125849 be interpreted meaningfully by a single thread. Thus, any system resources that might be
125850 referred to in data structures need to be shared between all threads. File descriptors, pathnames,
125851 and pointers to stack variables are all things that programmers want to share between their
125852 threads. Thus, the file descriptor table, the root directory, the current working directory, and the
125853 address space have to be shared.

125854 Library implementations are possible as long as the effective behavior is as if system services
125855 invoked by one thread do not suspend other threads. This may be difficult for some library
125856 implementations on systems that do not provide asynchronous facilities.

125857 See Section B.2.9 (on page 3806) for additional rationale.

125858 Austin Group Defect 792 is applied, changing this definition.

125859 **Thread ID**

125860 See Section B.2.9.2 (on page 3823) for additional rationale.

125861 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

125862 **Thread Lifetime**

125863 Austin Group Defect 792 is applied, adding this definition.

125864 **Thread Termination**

125865 Austin Group Defect 792 is applied, adding this definition.

125866 **Thread-Safe**

125867 All functions required by POSIX.1-2024 need to be thread-safe; see Section A.4.22 (on page 3687)
125868 and Section B.2.9.1 (on page 3820) for additional rationale.

125869 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

125870 **Thread-Specific Data Key**

125871 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

125872 **User Database**

125873 There are no references in POSIX.1-2024 to a ``passwd file'' or a ``group file'', and there is no
125874 requirement that the *group* or *passwd* databases be kept in files containing editable text. Many
125875 large timesharing systems use *passwd* databases that are hashed for speed. Certain security
125876 classifications prohibit certain information in the *passwd* database from being publicly readable.

125877 The term ``encoded'' is used instead of ``encrypted'' in order to avoid the implementation
125878 connotations (such as reversibility or use of a particular algorithm) of the latter term.

125879 The *getgrent*( ), *setgrent*( ), *endgrent*( ), *getpwent*( ), *setpwent*( ), and *endpwent*( ) functions are not
125880 included as part of the base standard because they provide a linear database search capability
125881 that is not generally useful (the *getpwuid*( ), *getpwnam*( ), *getgrgid*( ), and *getgrnam*( ) functions are
125882 provided for keyed lookup) and because in certain distributed systems, especially those with
125883 different authentication domains, it may not be possible or desirable to provide an application
125884 with the ability to browse the system databases indiscriminately. They are provided on XSI-
125885 conformant systems due to their historical usage by many existing applications.

125886 A change from historical implementations is that the structures used by these functions have
125887 fields of the types **gid_t** and **uid_t**, which are required to be defined in the **<sys/types.h>** header.
125888 POSIX.1-2024 requires implementations to ensure that these types are defined by inclusion of
125889 **<grp.h>** and **<pwd.h>**, respectively, without imposing any name space pollution or errors from
125890 redefinition of types.

125891 POSIX.1-2024 is silent about the content of the strings containing user or group names. These
125892 could be digit strings. POSIX.1-2024 is also silent as to whether such digit strings bear any
125893 relationship to the corresponding (numeric) user or group ID.

125894 *Database Access*

125895 The thread-safe versions of the user and group database access functions return values in user-
125896 supplied buffers instead of possibly using static data areas that may be overwritten by each call.

125897 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/11 is applied, removing the words ``of
125898 implementation-defined format''.

125899 **User ID**

125900 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0016 [511] is applied.

125901 **User Name**

125902 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0017 [584] is applied.

125903 **Virtual Processor***

125904 The term ``virtual processor'' was chosen as a neutral term describing all kernel-level
125905 schedulable entities, such as processes, Mach tasks, or lightweight processes. Implementing
125906 threads using multiple processes as virtual processors, or implementing multiplexed threads
125907 above a virtual processor layer, should be possible, provided some mechanism has also been
125908 implemented for sharing state between processes or virtual processors. Many systems may also
125909 wish to provide implementations of threads on systems providing ``shared processes'' or
125910 ``variable-weight processes''. It was felt that exposing such implementation details would
125911 severely limit the type of systems upon which the threads interface could be supported and
125912 prevent certain types of valid implementations. It was also determined that a virtual processor

125913 interface was out of the scope of the Rationale (Informative) volume of POSIX.1-2024.

125914 **White Space**

125915 Austin Group Defect 1163 is applied, clarifying the definition of white space and adding
125916 definitions of white-space byte, white-space character, and white-space wide character.

125917 **XSI**

125918 This is included to allow POSIX.1-2024 to be adopted as an IEEE standard and a standard of The
125919 Open Group, serving both POSIX and the Single UNIX Specification in a core set of volumes.

125920 When POSIX.1 and the Single UNIX Specification were merged, the term ``XSI'' had been used
125921 for over 10 years in connection with the XPG series and the first and second versions of the base
125922 volumes of the Single UNIX Specification. The XSI margin code was introduced to denote the
125923 extended or more restrictive semantics beyond POSIX that are applicable to UNIX systems.

125924 **Zombie Process**

125925 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0018 [690] is applied.

125926 **Zombie Thread**

125927 Austin Group Defect 792 is applied, adding this definition.

125928 A.4 General Concepts

125929 The general concepts are similar in nature to the definitions section, with the exception that a
125930 term defined in general concepts can contain normative requirements.

125931 A.4.1 Case Insensitive Comparisons

125932 Case-insensitive matching is defined in this standard in terms of a simple algorithm whereby, for
125933 each character in the string to be matched, if the character is uppercase then the lowercase
125934 equivalent (if any) is also checked for a match, and if the character is lowercase then the
125935 uppercase equivalent (if any) is also checked for a match. It is described this way to make the
125936 expected behavior easier to understand; however, implementations may internally use more
125937 sophisticated algorithms to improve efficiency, provided that the result is the same as the simple
125938 algorithm would produce.

125939 Austin Group Defect 1031 is applied, adding case insensitive comparisons.

125940 A.4.2 Concurrent Execution

125941 There is no additional rationale provided for this section.

A.4.3 Default Initialization

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0019 [934] is applied.

Austin Group Defect 940 is applied, removing text that was conditional on the all-zero bit pattern of a pointer object being a null pointer, as this is now mandated.

A.4.4 Directory Operations

Earlier versions of this standard did not make clear that directory modifications are performed atomically and serially, although that is the historical behavior and was always intended.

Austin Group Defect 672 is applied, adding this subsection.

A.4.5 Directory Protection

There is no additional rationale provided for this section.

A.4.6 Extended Security Controls

Allowing an implementation to define extended security controls enables the use of POSIX.1-2024 in environments that require different or more rigorous security than that provided in POSIX.1. Extensions are allowed in two areas: privilege and file access permissions. The semantics of these areas have been defined to permit extensions with reasonable, but not exact, compatibility with all existing practices. For example, the elimination of the superuser definition precludes identifying a process as privileged or not by virtue of its effective user ID.

A.4.7 File Access Permissions

A process should not try to anticipate the result of an attempt to access data by *a priori* use of these rules. Rather, it should make the attempt to access data and examine the return value (and possibly *errno* as well), or use *access*( ). An implementation may include other security mechanisms in addition to those specified in POSIX.1, and an access attempt may fail because of those additional mechanisms, even though it would succeed according to the rules given in this section. (For example, the user's security level might be lower than that of the object of the access attempt.) The supplementary group IDs provide another reason for a process to not attempt to anticipate the result of an access attempt.

Since the current standard does not specify a method for opening a directory for searching, it is unspecified whether search permission on the *fd* argument to *openat*( ) and related functions is based on whether the directory was opened with search mode or on the current permissions allowed by the directory at the time a search is performed. When there is existing practice that supports opening directories for searching, it is expected that these functions will be modified to specify that the search permissions will be granted based on the file access modes of the directory's file descriptor identified by *fd*, and not on the mode of the directory at the time the directory is searched.

A.4.8 File Hierarchy

Though the file hierarchy is commonly regarded to be a tree, POSIX.1 does not define it as such for three reasons:

1. Links may join branches.
2. In some network implementations, there may be no single absolute root directory; see *pathname resolution*.
3. With symbolic links, the file system need not be a tree or even a directed acyclic graph.

A.4.9 Filenames

Historically, certain filenames and pathnames have been reserved. This list includes **core**, **/etc/passwd**, and so on. Conforming applications should avoid these.

Most historical implementations prohibit case folding in filenames; that is, treating uppercase and lowercase alphabetic characters as identical. However, some consider case folding desirable:

- For user convenience
- For ease-of-implementation of the POSIX.1 interface as a hosted system on some popular operating systems

Variants, such as maintaining case distinctions in filenames, but ignoring them in comparisons, have been suggested. Methods of allowing escaped characters of the case opposite the default have been proposed.

Many reasons have been expressed for not allowing case folding, including:

- No solid evidence has been produced as to whether case-sensitivity or case-insensitivity is more convenient for users.
- Making case-insensitivity a POSIX.1 implementation option would be worse than either having it or not having it, because:
  - More confusion would be caused among users.
  - Application developers would have to account for both cases in their code.
  - POSIX.1 implementors would still have other problems with native file systems, such as short or otherwise constrained filenames or pathnames, and the lack of hierarchical directory structure.
- Case folding is not easily defined in many European languages, both because many of them use characters outside the US ASCII alphabetic set, and because:
  - In Spanish, the digraph `"ll"` is considered to be a single letter, the capitalized form of which may be either `"Ll"` or `"LL"`, depending on context.
  - In French, the capitalized form of a letter with an accent may or may not retain the accent, depending on the country in which it is written.
  - In German, the sharp ess may be represented as a single character resembling a Greek beta (*β*) in lowercase, but as the digraph `"SS"` in uppercase.
  - In Greek, there are several lowercase forms of some letters; the one to use depends on its position in the word. Arabic has similar rules.

126014 • Many East Asian languages, including Japanese, Chinese, and Korean, do not distinguish
126015 case and are sometimes encoded in character sets that use more than one byte per
126016 character.

126017 • Multiple character codes may be used on the same machine simultaneously. There are
126018 several ISO character sets for European alphabets. In Japan, several Japanese character
126019 codes are commonly used together, sometimes even in filenames; this is evidently also the
126020 case in China. To handle case insensitivity, the kernel would have to at least be able to
126021 distinguish for which character sets the concept made sense.

126022 • The file system implementation historically deals only with bytes, not with characters.
126023 Limitations on valid encodings ensure that the byte sequences for the <slash> character,
126024 <period> character, and <NUL> character will not be confused with any other character in
126025 any locale. However, there exist common single-shift encodings where other single-byte
126026 characters from the portable filename character set can also occur as a subset of a multi-
126027 byte character, making case folding of portable filename bytes dependent on the context of
126028 whether a shift-state is active.

126029 • The purpose of POSIX.1 is to standardize the common, existing definition, not to change it.
126030 Mandating case-insensitivity would make all historical implementations non-standard.

126031 • Not only the interface, but also application programs would need to change, counter to the
126032 purpose of having minimal changes to existing application code.

126033 • At least one of the original developers of the UNIX system has expressed objection in the
126034 strongest terms to either requiring case-insensitivity or making it an option, mostly on the
126035 basis that POSIX.1 should not hinder portability of application programs across related
126036 implementations in order to allow compatibility with unrelated operating systems.

126037 Two proposals were entertained regarding case folding in filenames:

126038 1. Remove all wording that previously permitted case folding.

126039 Rationale Case folding is inconsistent with the portable filename character set and
126040 filename definitions (all bytes except <slash> and null). No known
126041 implementations allowing all bytes except <slash> and null also do case
126042 folding.

126043 2. Change ``though this practice is not recommended:'' to ``although this practice is strongly
126044 discouraged.''

126045 Rationale If case folding must be included in POSIX.1, the wording should be stronger
126046 to discourage the practice.

126047 The consensus selected the first proposal. Otherwise, a conforming application would have to
126048 assume that case folding would occur when it was not wanted, but that it would not occur when
126049 it was wanted.

126050 ### A.4.10 Filename Portability

126051 Filenames should be constructed from the portable filename character set because the use of
126052 other characters can be confusing or ambiguous in certain contexts. (For example, the use of a
126053 <colon> (`':'`) in a pathname could cause ambiguity if that pathname were included in a *PATH*
126054 definition.)

126055 The constraint on use of the <hyphen-minus> character as the first character of a portable
126056 filename is a constraint on application behavior and not on implementations, since applications
126057 might not work as expected when such a filename is passed as a command line argument.

126058 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0020 [584] is applied.

126059 A.4.11 File System Cache

126060 Earlier versions of this standard did not specify the behavior of *aio_fsync*(), *fdatasync*(), or
126061 *fsync*() on directories, nor did they specify constraints on the underlying storage in the absence
126062 of calls to *aio_fsync*(), *fdatasync*(), or *fsync*().

126063 Although directory operations are atomic and serializable, they are not necessarily durable. An
126064 application that requires a directory modification to be durable should use *fdatasync*() or *fsync*()
126065 (or *aio_fsync*()) on the directory. However, the intention of the requirements for directory
126066 modifications is that most applications should not need to do this. For example, a common
126067 method of updating a file is to create a new temporary file, call *fdatasync*() or *fsync*() to
126068 synchronize the new file, and then use *rename*() to replace the old file with the new file. If a
126069 crash occurs after the *rename*(), then the file being updated will have either its old contents or its
126070 new contents on the storage device when the system is rebooted. An application needs to
126071 synchronize the directory only if it wants to be sure the updated file will have its new contents
126072 on the storage device.

126073 Some operations, such as *rename*(), can affect more than one directory, whereas synchronization
126074 calls such as *fsync*() can affect at most one directory at a time. Two calls to *fsync*() may be
126075 needed after a *rename*() to ensure its durability.

126076 If the file system is inconsistent after a crash it is usually automatically checked and repaired
126077 when the system is rebooted, or can be repaired manually using a utility such as *fsck*.

126078 If an unrecoverable I/O error occurs when cache is transferred to storage, this standard provides
126079 no way for applications to discover the error reliably. Implementations are encouraged to report
126080 such errors on subsequent reads of the storage.

126081 Austin Group Defect 672 is applied, adding this subsection.

126082 A.4.12 File Times Update

126083 This section reflects the actions of historical implementations. The times are not updated
126084 immediately, but are only marked for update by the functions. An implementation may update
126085 these times immediately.

126086 The accuracy of the time update values is intentionally left unspecified so that systems can
126087 control the bandwidth of a possible covert channel.

126088 The wording was carefully chosen to make it clear that there is no requirement that the
126089 conformance document contain information that might incidentally affect file timestamps. Any
126090 function that performs pathname resolution might update several last data access timestamps.
126091 Functions such as *getpwnam*() and *getgrnam*() might update the last data access timestamp of
126092 some specific file or files. It is intended that these are not required to be documented in the
126093 conformance document, but they should appear in the system documentation.

126094 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0021 [626] is applied.

126095 ### A.4.13 Host and Network Byte Order

126096 There is no additional rationale provided for this section.

126097 ### A.4.14 Measurement of Execution Time

126098 The methods used to measure the execution time of processes and threads, and the precision of
126099 these measurements, may vary considerably depending on the software architecture of the
126100 implementation, and on the underlying hardware. Implementations can also make tradeoffs
126101 between the scheduling overhead and the precision of the execution time measurements.
126102 POSIX.1-2024 does not impose any requirement on the accuracy of the execution time; it instead
126103 specifies that the measurement mechanism and its precision are implementation-defined.

126104 ### A.4.15 Memory Ordering and Synchronization

126105 Austin Group Defect 1302 is applied, adding the *Memory Ordering* subsection, adapted from the
126106 ISO/IEC 9899:2018 standard.

126107 #### *A.4.15.1 Memory Ordering*

126108 There is no additional rationale provided for this section.

126109 #### *A.4.15.2 Memory Synchronization*

126110 In older multi-processors, access to memory by the processors was strictly multiplexed. This
126111 meant that a processor executing program code interrogates or modifies memory in the order
126112 specified by the code and that all the memory operation of all the processors in the system
126113 appear to happen in some global order, though the operation histories of different processors are
126114 interleaved arbitrarily. The memory operations of such machines are said to be sequentially
126115 consistent. In this environment, threads can synchronize using ordinary memory operations. For
126116 example, a producer thread and a consumer thread can synchronize access to a circular data
126117 buffer as follows:

```
126118 int rdptr = 0;
126119 int wrptr = 0;
126120 data_t buf[BUFSIZE];

126121 Thread 1:
126122     while (work_to_do) {
126123         int next;

126124         buf[wrptr] = produce();
126125         next = (wrptr + 1) % BUFSIZE;
126126         while (rdptr == next)
126127             ;
126128         wrptr = next;
126129 }

126130 Thread 2:
126131     while (work_to_do) {
126132         while (rdptr == wrptr)
126133             ;
126134         consume(buf[rdptr]);
```



```
            rdptr = (rdptr + 1) % BUFSIZE;
        }
```

In modern multi-processors, these conditions are relaxed to achieve greater performance. If one processor stores values in location A and then location B, then other processors loading data from location B and then location A may see the new value of B but the old value of A. The memory operations of such machines are said to be weakly ordered. On these machines, the circular buffer technique shown in the example will fail because the consumer may see the new value of *wrptr* but the old value of the data in the buffer. In such machines, synchronization can only be achieved through the use of special instructions that enforce an order on memory operations. Most high-level language compilers only generate ordinary memory operations to take advantage of the increased performance. They usually cannot determine when memory operation order is important and generate the special ordering instructions. Instead, they rely on the programmer to use synchronization primitives correctly to ensure that modifications to a location in memory are ordered with respect to modifications and/or access to the same location in other threads. Access to read-only data need not be synchronized. The resulting program is said to be data race-free.

Synchronization is still important even when accessing a single primitive variable (for example, an integer). On machines where the integer may not be aligned to the bus data width or be larger than the data width, a single memory load may require multiple memory cycles. This means that it may be possible for some parts of the integer to have an old value while other parts have a newer value. On some processor architectures this cannot happen, but portable programs cannot rely on this.

In summary, a portable multi-threaded program, or a multi-process program that shares writable memory between processes, has to use the synchronization primitives to synchronize data access. It cannot rely on modifications to memory being observed by other threads in the order written in the application or even on modification of a single variable being seen atomically.

Conforming applications may only use the functions listed to synchronize threads of control with respect to memory access. There are many other candidates for functions that might also be used. Examples are: signal sending and reception, or pipe writing and reading. In general, any function that allows one thread of control to wait for an action caused by another thread of control is a candidate. POSIX.1-2024 does not require these additional functions to synchronize memory access since this would imply the following:

- All these functions would have to be recognized by advanced compilation systems so that memory operations and calls to these functions are not reordered by optimization.
- All these functions would potentially have to have memory synchronization instructions added, depending on the particular machine.
- The additional functions complicate the model of how memory is synchronized and make automatic data race detection techniques impractical.

Formal definitions of the memory model were rejected as unreadable by the vast majority of programmers. In addition, most of the formal work in the literature has concentrated on the memory as provided by the hardware as opposed to the application programmer through the compiler and runtime system. It was believed that a simple statement intuitive to most programmers would be most effective. POSIX.1-2024 defines functions that can be used to synchronize access to memory, but it leaves open exactly how one relates those functions to the semantics of each function as specified elsewhere in POSIX.1-2024. POSIX.1-2024 also does not make a formal specification of the partial ordering in time that the functions can impose, as that is implied in the description of the semantics of each function. It simply states that the programmer has to ensure that modifications do not occur ``simultaneously'' with other access

126184 to a memory location.

126185 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/4 is applied, adding a new paragraph
126186 beneath the table of functions: ``The *pthread_once*( ) function shall synchronize memory for the
126187 first call in each thread for a given **pthread_once_t** object.''.

126188 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0022 [863] is applied.

126189 Austin Group Defect 1216 is applied, adding *pthread_cond_clockwait*( ), *pthread_mutex_clocklock*( ),
126190 *pthread_rwlock_clockrdlock*( ), *pthread_rwlock_clockwrlock*( ), and *sem_clockwait*( ) to the list of
126191 functions that synchronize memory.

126192 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899: 2018 standard.

126193 Austin Group Defect 1426 is applied, clarifying under what conditions the functions named in
126194 this section are required to synchronize memory, and adding *pthread_mutex_setprioceiling*( ) to the
126195 named functions.

126196 Austin Group Defect 1625 is applied, adding *waitid*( ) to the list of functions that synchronize
126197 memory with respect to other threads on all successful calls.

126198 ### A.4.16 Pathname Resolution

126199 It is necessary to differentiate between the definition of pathname and the concept of pathname
126200 resolution with respect to the handling of trailing <slash> characters. By specifying the behavior
126201 here, it is not possible to provide an implementation that is conforming but extends all interfaces
126202 that handle pathnames to also handle strings that are not legal pathnames (because they have
126203 trailing <slash> characters).

126204 Pathnames that end with one or more trailing <slash> characters must refer to directory paths.
126205 Earlier versions of this standard were not specific about the distinction between trailing <slash>
126206 characters on files and directories, and both were permitted.

126207 Two types of implementation have been prevalent; those that ignored trailing <slash> characters
126208 on all pathnames regardless, and those that permitted them only on existing directories.

126209 An earlier version of this standard required that a pathname with a trailing <slash> character be
126210 treated as if it had a trailing `"/."` everywhere. This specification was ambiguous. In situations
126211 where the intent was that the application wanted to require the implementation to accept the
126212 pathname only if it named a directory (existing or to be created as a result of the call performing
126213 pathname resolution), literally adding a `'.'` after the trailing <slash> could be interpreted to
126214 require use of that pathname to fail. Some of the uses that created ambiguous requirements
126215 included *mkdir*("*newdir*/") and *rmdir*("*existing-dir*/"). POSIX.1-2024 requires that a pathname
126216 with a trailing <slash> be rejected unless it refers to a file that is a directory or to a file that is to
126217 be created as a directory. The *rename*( ) function and the *mv* utility further specify that a trailing
126218 <slash> cannot be used on a pathname naming a file that does not exist when used as the last
126219 argument to *rename*( ) or *renameat*( ), or as the last operand to *mv*.

126220 Note that this change does not break any conforming applications; since there were two different
126221 types of implementation, no application could have portably depended on either behavior. This
126222 change does however require some implementations to be altered to remain compliant.
126223 Substantial discussion over a three-year period has shown that the benefits to application
126224 developers outweighs the disadvantages for some vendors.

126225 On a historical note, some early applications automatically appended a `'/'` to every path.
126226 Rather than fix the applications, the system implementation was modified to accept this
126227 behavior by ignoring any trailing <slash>.

126228 Each directory has exactly one parent directory which is represented by the name **dot-dot** in the
126229 first directory. No other directory, regardless of linkages established by symbolic links, is
126230 considered the parent directory by POSIX.1-2024.

126231 There are two general categories of interfaces involving pathname resolution: those that follow
126232 the symbolic link, and those that do not. There are several exceptions to this rule; for example,
126233 *open*(*path*,O_CREAT|O_EXCL) will fail when *path* names a symbolic link. However, in all other
126234 situations, the *open*( ) function will follow the link.

126235 What the filename **dot-dot** refers to relative to the root directory is implementation-defined. In
126236 Version 7 it refers to the root directory itself; this is the behavior mentioned in POSIX.1-2024. In
126237 some networked systems the construction **/../hostname/** is used to refer to the root directory of
126238 another host, and POSIX.1 permits this behavior.

126239 Other networked systems use the construct **//hostname** for the same purpose; that is, a double
126240 initial <slash> is used. There is a potential problem with existing applications that create full
126241 pathnames by taking a trunk and a relative pathname and making them into a single string
126242 separated by `'/'`, because they can accidentally create networked pathnames when the trunk is
126243 `'/'`. This practice is not prohibited because such applications can be made to conform by
126244 simply changing to use `"//"` as a separator instead of `'/'`:

126245 • If the trunk is `'/'`, the full pathname will begin with `"///"` (the initial `'/'` and the
126246 separator `"//"`). This is the same as `'/'`, which is what is desired. (This is the general
126247 case of making a relative pathname into an absolute one by prefixing with `"///"` instead
126248 of `'/'`.)

126249 • If the trunk is `"/A"`, the result is `"/A//..."`; since non-leading sequences of two or more
126250 <slash> characters are treated as a single <slash>, this is equivalent to the desired
126251 `"/A/..."`.

126252 • If the trunk is `"//A"`, the implementation-defined semantics will apply. (The multiple
126253 <slash> rule would apply.)

126254 Application developers should avoid generating pathnames that start with `"//"`.
126255 Implementations are strongly encouraged to avoid using this special interpretation since a
126256 number of applications currently do not follow this practice and may inadvertently generate
126257 `"//..."`.

126258 The term ``root directory'' is only defined in POSIX.1 relative to the process. In some
126259 implementations, there may be no absolute root directory. The initialization of the root directory
126260 of a process is implementation-defined.

126261 When the standard says: ``Pathname resolution for a given pathname shall yield the same results
126262 when used by any interface in POSIX.1-2024 as long as there are no changes to any files
126263 evaluated during pathname resolution for the given pathname between resolutions'', this
126264 applies to absolute pathnames or to relative pathnames from the same current working
126265 directory. Using the same relative pathname from two different working directories may yield
126266 different results.

126267 Earlier versions of this standard were unclear as to whether a pathname was required to be a
126268 character string or just a string. This standard is now clear that filenames are just strings, and
126269 that pathname processing is locale-independent.

126270 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0023 [541,649,825] and
126271 XBD/TC2-2008/0024 [825] are applied.

126272 Austin Group Defect 1603 is applied, making a wording improvement related to symbolic links
126273 to directories.

A.4.17 Process ID Reuse

There is no additional rationale provided for this section.

A.4.18 Scheduling Policy

There is no additional rationale provided for this section.

A.4.19 Seconds Since the Epoch

Coordinated Universal Time (UTC) includes leap seconds. However, in POSIX time (seconds since the Epoch), leap seconds are ignored (not applied) to provide an easy and compatible method of computing time differences. Broken-down POSIX time is therefore not necessarily UTC, despite its appearance.

As of December 2007, 23 leap seconds had been added to UTC since the Epoch, 1 January, 1970. Historically, one leap second is added every 15 months on average, so this offset can be expected to grow with time.

Most systems' notion of ``time'' is that of a continuously increasing value, so this value should increase even during leap seconds. However, not only do most systems not keep track of leap seconds, but most systems are probably not synchronized to any standard time reference. Therefore, it is inappropriate to require that a time represented as seconds since the Epoch precisely represent the number of seconds between the referenced time and the Epoch.

It is sufficient to require that applications be allowed to treat this time as if it represented the number of seconds between the referenced time and the Epoch. It is the responsibility of the vendor of the system, and the administrator of the system, to ensure that this value represents the number of seconds between the referenced time and the Epoch as closely as necessary for the application being run on that system.

It is important that the interpretation of time names and seconds since the Epoch values be consistent across conforming systems; that is, it is important that all conforming systems interpret ``536 457 599 seconds since the Epoch'' as 59 seconds, 59 minutes, 23 hours 31 December 1986, regardless of the accuracy of the system's idea of the current time. The expression is given to ensure a consistent interpretation, not to attempt to specify the calendar. The relationship between *tm_yday* and the day of week, day of month, and month is in accordance with the Gregorian calendar, and so is not specified in POSIX.1.

Consistent interpretation of seconds since the Epoch can be critical to certain types of distributed applications that rely on such timestamps to synchronize events. The accrual of leap seconds in a time standard is not predictable. The number of leap seconds since the Epoch will likely increase. POSIX.1 is more concerned about the synchronization of time between applications of astronomically short duration.

Note that *tm_yday* is zero-based, not one-based, so the day number in the example above is 364. Note also that the division is an integer division (discarding remainder) as in the C language.

Note also that the meaning of *gmtime*(), *localtime*(), and *mktime*() is specified in terms of this expression. However, the ISO C standard computes *tm_yday* from *tm_mday*, *tm_mon*, and *tm_year* in *mktime*(). Because it is stated as a (bidirectional) relationship, not a function, and because the conversion between month-day-year and day-of-year dates is presumed well known and is also a relationship, this is not a problem.

The number of seconds since the epoch overflows a signed 32-bit integer in 2038. This standard requires that **time_t** is an integer type with a width of at least 64 bits (in conforming

programming environments). The requirement that **time_t** is an integer type is an additional constraint beyond the ISO C standard, which allows a real-floating **time_t**. Implementation practice has shown that much existing code is unprepared to deal with a floating-point **time_t**, and that use of **struct timespec** is a more uniform way to provide sub-second time manipulation within applications.

See also Epoch (on page 3653).

The topic of whether seconds since the Epoch should account for leap seconds has been debated on a number of occasions, and each time consensus was reached (with acknowledged dissent each time) that the majority of users are best served by treating all days identically. (That is, the majority of applications were judged to assume a single length—as measured in seconds since the Epoch—for all days. Thus, leap seconds are not applied to seconds since the Epoch.) Those applications which do care about leap seconds can determine how to handle them in whatever way those applications feel is best. This was particularly emphasized because there was disagreement about what the best way of handling leap seconds might be. It is a practical impossibility to mandate that a conforming implementation must have a fixed relationship to any particular official clock (consider isolated systems, or systems performing ``reruns'' by setting the clock to some arbitrary time).

Note that as a practical consequence of this, the length of a second as measured by some external standard is not specified. This unspecified second is nominally equal to an International System (SI) second in duration. Applications must be matched to a system that provides the particular handling of external time in the way required by the application.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/12 is applied, making an editorial correction to the paragraph commencing ``How any changes to the value of seconds ...''.

Austin Group Defect 1627 is applied, clarifying that the relationship between the actual date and time in Coordinated Universal Time, as determined by the International Earth Rotation Service, and the system's current value for seconds since the Epoch is unspecified.

A.4.20 Semaphore

Austin Group Defect 502 is applied, clarifying the range of values that an XSI semaphore can have.

Austin Group Defect 1116 is applied, removing a reference to the Semaphores option that existed in earlier versions of this standard.

A.4.21 Special Device Drivers

POSIX systems interact with their physical environment using a variety of devices (such as analog-digital converters, digital-analog converters, counters, and video graphic equipment), which provide a set of services that cannot be fully utilized in terms of read and/or write semantics. Traditional practice uses a single function, called *ioctl*( ), to encapsulate all the control operations on the different devices connected to the system, both special or common devices. The POSIX.1-1988 standard developers decided not to standardize this interface because it was not type safe, it had a variable number of parameters, and it had behaviors that could not be specified by the standard because they were driver-dependent. Instead, the POSIX.1-1988 standard defined a device-specific application program interface (API) for a common class of drivers, Terminals. Later, The Single UNIX Specification, Version 1 included the *ioctl*( ) function, but restricted it to control of STREAMS devices.

Although the POSIX.1-1988 standard's solution for common classes of devices is the best from

the point of view of application portability, there is still a need for a way to interact with special, or even common devices, for which developing a full standard API is not practical. The device control option standardized in POSIX.26 and now included in this standard is a general method for interfacing to the widest possible range of devices, through a new service to pass control information and commands between the application and the device drivers.

A driver for a special device will normally not be portable between POSIX implementations, but an application that uses such a driver can be made portable if all functions calling the driver are well defined and standardized. Users and integrators of realtime systems often add drivers for special devices, and a standardized function format for interfacing with these devices greatly simplifies this process.

Austin Group Defect 729 is applied, adding this subsection.

A.4.22 Thread-Safety

Where the interface of a function required by POSIX.1-2024 precludes thread-safety, an alternate thread-safe form is provided. The names of these thread-safe forms are the same as the non-thread-safe forms with the addition of the suffix ``_r''. The suffix ``_r'' is historical, where the `'r'` stood for ``reentrant''.

In some cases, thread-safety is provided by restricting the arguments to an existing function.

See also Section B.2.9.1 (on page 3820).

A.4.23 Treatment of Error Conditions for Mathematical Functions

It is intended that undeserved underflow and inexact floating-point exceptions are raised only if avoiding them would be too costly.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0025 [543] is applied.

Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899: 2018 standard.

A.4.24 Treatment of NaN Arguments for Mathematical Functions

There is no additional rationale provided for this section.

A.4.25 Utility

There is no additional rationale provided for this section.

A.4.26 Variable Assignment

Austin Group Defect 351 is applied, adding a requirement relating to declaration utilities.

A.5 File Format Notation

The notation for spaces allows some flexibility for application output. Note that an empty character position in *format* represents one or more <blank> characters on the output (not *white space*, which can include <newline> characters). Therefore, another utility that reads that output as its input must be prepared to parse the data using *scanf*( ), *awk*, and so on. The `'Δ'` character is used when exactly one <space> is output.

The treatment of integers and spaces is different from the *printf*( ) function in that they can be surrounded with <blank> characters. This was done so that, given a format such as:

`"%d\n",`<*foo*>

the implementation could use a *printf*( ) call such as:

```
printf("%6d\n", foo);
```

and still conform. This notation is thus somewhat like *scanf*( ) in addition to *printf*( ).

The *printf*( ) function was chosen as a model because most of the standard developers were familiar with it. One difference from the C function *printf*( ) is that the `l` and `h` conversion specifier characters are not used. As expressed by the Shell and Utilities volume of POSIX.1-2024, there is no differentiation between decimal values for type **int**, type **long**, or type **short**. The conversion specifications `%d` or `%i` should be interpreted as an arbitrary length sequence of digits. Also, no distinction is made between single precision and double precision numbers (**float** or **double** in C). These are simply referred to as floating-point numbers.

Many of the output descriptions in the Shell and Utilities volume of POSIX.1-2024 use the term ``line'', such as:

```
"%s", <input line>
```

Since the definition of *line* includes the trailing <newline> already, there is no need to include a `'\n'` in the format; a double <newline> would otherwise result.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0026 [584] is applied.

Austin Group Defect 1205 is applied, changing the description of the `%` conversion specifier.

Austin Group Defect 1687 is applied, clarifying the references to <blank> characters to specify they are from the portable character set.

A.6 Character Set

A.6.1 Portable Character Set

The portable character set is listed in full so there is no dependency on the ISO/IEC 646:1991 standard (or historically ASCII) encoded character set, although the set is identical to the characters defined in the International Reference version of the ISO/IEC 646:1991 standard.

POSIX.1-2024 poses no requirement that multiple character sets or codesets be supported, leaving this as a marketing differentiation for implementors. Although multiple charmap files are supported, it is the responsibility of the implementation to provide the file(s); if only one is provided, only that one will be accessible using the *localedef* **–f** option.

The statement about invariance in codesets for the portable character set is worded to avoid

precluding implementations where multiple incompatible codesets are available (for instance, ASCII and EBCDIC). The standard utilities cannot be expected to produce predictable results if they access portable characters that vary on the same implementation.

Not all character sets need include the portable character set, but each locale must include it. For example, a Japanese-based locale might be supported by a mixture of character sets: JIS X 0201 Roman (a Japanese version of the ISO/IEC 646:1991 standard), JIS X 0208, and JIS X 0201 Katakana. Not all of these character sets include the portable characters, but at least one does (JIS X 0201 Roman).

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0027 [584,967] and XBD/TC2-2008/0028 [745] are applied.

A.6.2 Character Encoding

Encoding mechanisms based on single shifts, such as the EUC encoding used in some Asian and other countries, can be supported via the current charmap mechanism. With single-shift encoding, each character is preceded by a shift code (SS2 or SS3). A complete EUC code, consisting of the portable character set (G0) and up to three additional character sets (G1, G2, G3), can be described using the current charmap mechanism; the encoding for each character in additional character sets G2 and G3 must then include their single-shift code. Other mechanisms to support locales based on encoding mechanisms such as locking shift are not addressed by this volume of POSIX.1-2024.

The encodings for <slash> and <period> are required to be the same across all locales, in part because pathname resolution requires recognition of these bytes. It is a fortunate accident that all common shift-based encodings did not use either <slash> or <period> as a valid second byte in a multi-byte character.

The encodings for <newline> and <carriage-return> are required to be the same across all locales since they are special to the general terminal interface and cannot be changed (see XBD Section 11.1.9, on page 203).

Earlier versions of this standard did not state the requirement that the POSIX locale contains 256 single-byte characters. This was an oversight; the intention was always that the POSIX locale should have an 8-bit-clean single-byte encoding.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0029 [663,967] and XBD/TC2-2008/0030 [745] are applied.

A.6.3 C Language Wide-Character Codes

The standard does not specify how wide characters are encoded or provide a method for defining wide characters in a charmap. It specifies ways of translating between wide characters and multi-byte characters. The standard does not prevent an extension from providing a method to define wide characters.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/13 is applied, adding a statement that the standard has no means of defining a wide-character codeset.

Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

A.6.4 Character Set Description File

IEEE PASC Interpretation 1003.2 #196 is applied, removing three lines of text dealing with ranges of symbolic names using position constant values which had been erroneously included in the final IEEE P1003.2b draft standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/14 is applied, correcting the example and adding a statement that the standard provides no means of defining a wide-character codeset.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/15 is applied, allowing the value zero for the width value of **WIDTH** and **WIDTH_DEFAULT**. This is required to cover some existing locales.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0031 [967] is applied.

A.6.4.1 State-Dependent Character Encodings

A requirement was considered that would force utilities to eliminate any redundant locking shifts, but this was left as a quality of implementation issue.

This change satisfies the following requirement from the ISO POSIX-2: 1993 standard, Annex H.1:

> *The support of state-dependent (shift encoding) character sets should be addressed fully. See descriptions of these in XBD Section 6.2 (on page 120). If such character encodings are supported, it is expected that this will impact XBD Section 6.2 (on page 120), Chapter 7 (on page 127), Chapter 9 (on page 179), and the comm, cut, diff, grep, head, join, paste, and tail utilities.*

The character set description file provides:

- The capability to describe character set attributes (such as collation order or character classes) independent of character set encoding, and using only the characters in the portable character set. This makes it possible to create generic *localedef* source files for all codesets that share the portable character set (such as the ISO 8859 family or IBM Extended ASCII).
- Standardized symbolic names for all characters in the portable character set, making it possible to refer to any such character regardless of encoding.

Implementations are free to choose their own symbolic names, as long as the names identified by the Base Definitions volume of POSIX.1-2024 are also defined; this provides support for already existing ``character names''.

The names selected for the members of the portable character set follow the ISO/IEC 8859-1: 1998 standard and the ISO/IEC 10646: 2020 standard. However, several commonly used UNIX system names occur as synonyms in the list:

- The historical UNIX system names are used for control characters.
- The word ``slash'' is given in addition to ``solidus''.
- The word ``backslash'' is given in addition to ``reverse-solidus''.
- The word ``hyphen'' is given in addition to ``hyphen-minus''.
- The word ``period'' is given in addition to ``full-stop''.
- For digits, the word ``digit'' is eliminated.

- For letters, the words ``Latin Capital Letter'' and ``Latin Small Letter'' are eliminated.
- The words ``left brace'' and ``right brace'' are given in addition to ``left-curly-bracket'' and ``right-curly-bracket''.
- The names of the digits are preferred over the numbers to avoid possible confusion between `'0'` and `'O'`, and between `'1'` and `'l'` (one and the letter ell).

The names for the control characters in XBD Chapter 6 (on page 117) were taken from the ISO/IEC 4873:1991 standard.

The charmap file was introduced to resolve problems with the portability of, especially, *localedef* sources. POSIX.1-2024 assumes that the portable character set is constant across all locales, but does not prohibit implementations from supporting two incompatible codings, such as both ASCII and EBCDIC. Such dual-support implementations should have all charmaps and *localedef* sources encoded using one portable character set, in effect cross-compiling for the other environment. Naturally, charmaps (and *localedef* sources) are only portable without transformation between systems using the same encodings for the portable character set. They can, however, be transformed between two sets using only a subset of the actual characters (the portable character set). However, the particular coded character set used for an application or an implementation does not necessarily imply different characteristics or collation; on the contrary, these attributes should in many cases be identical, regardless of codeset. The charmap provides the capability to define a common locale definition for multiple codesets (the same *localedef* source can be used for codesets with different extended characters; the ability in the charmap to define empty names allows for characters missing in certain codesets).

The **<escape_char>** declaration was added at the request of the international community to ease the creation of portable charmap files on terminals not implementing the default <backslash>-escape. The **<comment_char>** declaration was added at the request of the international community to eliminate the potential confusion between the <number-sign> and the hash sign.

The octal number notation with no leading zero required was selected to match those of *awk* and *tr* and is consistent with that used by *localedef*. To avoid confusion between an octal constant and the back-references used in *localedef* source, the octal, hexadecimal, and decimal constants must contain at least two digits. As single-digit constants are relatively rare, this should not impose any significant hardship. Provision is made for more digits to account for systems in which the byte size is larger than 8 bits. For example, a Unicode (ISO/IEC 10646:2020 standard) system that has defined 16-bit bytes may require six octal, four hexadecimal, and five decimal digits.

The decimal notation is supported because some newer international standards define character values in decimal, rather than in the old column/row notation.

The charmap identifies the coded character sets supported by an implementation. At least one charmap must be provided, but no implementation is required to provide more than one. Likewise, implementations can allow users to generate new charmaps (for instance, for a new version of the ISO 8859 family of coded character sets), but does not have to do so. If users are allowed to create new charmaps, the system documentation describes the rules that apply (for instance, ``only coded character sets that are supersets of the ISO/IEC 646:1991 standard IRV, no multi-byte characters'').

This addition of the **WIDTH** specification satisfies the following requirement from the ISO POSIX-2:1993 standard, Annex H.1:

> (9) *The definition of column position relies on the implementation's knowledge of the integral width of the characters. The charmap or LC_CTYPE locale definitions should be enhanced to allow application specification of these widths.*

The character ``width'' information was first considered for inclusion under *LC_CTYPE* but was moved because it is more closely associated with the information in the charmap than information in the locale source (cultural conventions information). Concerns were raised that formalizing this type of information is moving the locale source definition from the codeset-independent entity that it was designed to be to a repository of codeset-specific information. A similar issue occurred with the **<code_set_name>**, **<mb_cur_max>**, and **<mb_cur_min>** information, which was resolved to reside in the charmap definition.

The width definition was added to the IEEE P1003.2b draft standard with the intent that the *wcswidth*( ) and/or *wcwidth*( ) functions (currently specified in the System Interfaces volume of POSIX.1-2024) be the mechanism to retrieve the character width information.

A.7 Locale

A.7.1 General

The description of locales is based on work performed in the UniForum Technical Committee, Subcommittee on Internationalization. Wherever appropriate, keywords are taken from the ISO C standard or the X/Open Portability Guide.

The value used to specify a locale with environment variables is the name specified as the *name* operand to the *localedef* utility when the locale was created. This provides a verifiable method to create and invoke a locale.

The ``object'' definitions need not be portable, as long as ``source'' definitions are. Strictly speaking, source definitions are portable only between implementations using the same character set(s). Such source definitions, if they use symbolic names only, easily can be ported between systems using different codesets, as long as the characters in the portable character set (see XBD Section 6.1, on page 117) have common values between the codesets; this is frequently the case in historical implementations. Of source, this requires that the symbolic names used for characters outside the portable character set be identical between character sets. The definition of symbolic names for characters is outside the scope of POSIX.1-2024, but is certainly within the scope of other standards organizations.

Applications can select the desired locale by invoking the *setlocale*( ) function (or equivalent) with the appropriate value. If the function is invoked with an empty string, the value of the corresponding environment variable is used. If the environment variable is not set or is set to the empty string, the implementation sets the appropriate environment as defined in XBD Chapter 8 (on page 167).

The locale settings of individual categories cannot be truly independent and still guarantee correct results. For example, when collating two strings, characters must first be extracted from each string (governed by *LC_CTYPE*) before being mapped to collating elements (governed by *LC_COLLATE*) for comparison. That is, if *LC_CTYPE* is causing parsing according to the rules of a large, multi-byte code set (potentially returning 20 000 or more distinct character codeset values), but *LC_COLLATE* is set to handle only an 8-bit codeset with 256 distinct characters, meaningful results are obviously impossible.

Earlier versions of this standard stated that if different character sets are used by the locale categories, the results achieved by an application utilizing these categories are undefined. This was felt to be overly restrictive. For example, when setting:

```
LANG=en_US.utf8
```


126596 `LC_TIME=POSIX`

126597 on a system where the codeset for the POSIX locale is ASCII and the codeset for en_US.utf8 is
126598 UTF-8, all of the characters used in the *LC_TIME* locale data exist, with the same encoding, in
126599 the codeset used for *LC_CTYPE* (via *LANG*), so there is no reason for the behavior to be
126600 undefined in this case. This standard now has more precise requirements in this area.

126601 Austin Group Defect 1122 is applied, adding item 3 to the list of ways to select the locale to be
126602 used by some C-language functions.

126603 Austin Group Defect 1477 is applied, clarifying the behavior when locale categories have
126604 different character sets.

126605 ### A.7.2 POSIX Locale

126606 On POSIX.1 implementations the POSIX locale is equal to the C locale, even though the
126607 requirements for the POSIX locale are more extensive than the ISO C standard requirements for
126608 the C locale. To avoid being classified as a C-language function, the name has been changed to
126609 the POSIX locale; the environment variable value can be either `"POSIX"` or, for historical
126610 reasons, `"C"`.

126611 The POSIX definitions mirror the historical UNIX system behavior.

126612 The use of symbolic names for characters in the tables does not imply that the POSIX locale must
126613 be described using symbolic character names, but merely that it may be advantageous to do so.

126614 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0032 [796] and XBD/TC2-2008/0033
126615 [663] are applied.

126616 ### A.7.3 Locale Definition

126617 The decision to separate the file format from the *localedef* utility description was only partially
126618 editorial. Implementations may provide other interfaces than *localedef*. Requirements on ``the
126619 utility'', mostly concerning error messages, are described in this way because they are meant to
126620 affect the other interfaces implementations may provide as well as *localedef*.

126621 The text about POSIX2_LOCALEDEF does not mean that internationalization is optional; only
126622 that the functionality of the *localedef* utility is. REs, for instance, must still be able to recognize,
126623 for example, character class expressions such as `"[[:alpha:]]"`. A possible analogy is with
126624 an applications development environment; while all conforming implementations must be
126625 capable of executing applications, not all need to have the development environment installed.
126626 The assumption is that the capability to modify the behavior of utilities (and applications) via
126627 locale settings must be supported. If the *localedef* utility is not present, then the only choice is to
126628 select an existing (presumably implementation-documented) locale. An implementation could,
126629 for example, choose to support only the POSIX locale, which would in effect limit the amount of
126630 changes from historical implementations quite drastically. The *localedef* utility is still required,
126631 but would always terminate with an exit code indicating that no locale could be created.
126632 Supported locales must be documented using the syntax defined in this chapter. (This ensures
126633 that users can accurately determine what capabilities are provided. If the implementation
126634 decides to provide additional capabilities to the ones in this chapter, that is already provided
126635 for.)

126636 If the option is present (that is, locales can be created), then the *localedef* utility must be capable
126637 of creating locales based on the syntax and rules defined in this chapter. This does not mean that
126638 the implementation cannot also provide alternate means for creating locales.

The octal, decimal, and hexadecimal notations are the same employed by the charmap facility (see XBD Section 6.4, on page 121). To avoid confusion between an octal constant and a back-reference, the octal, hexadecimal, and decimal constants must contain at least two digits. As single-digit constants are relatively rare, this should not impose any significant hardship. Provision is made for more digits to account for systems in which the byte size is larger than 8 bits. For example, a Unicode (see the ISO/IEC 10646:2020 standard) system that has defined 16-bit bytes may require six octal, four hexadecimal, and five decimal digits. As with the charmap file, multi-byte characters are described in the locale definition file using ``big-endian'' notation for reasons of portability. There is no requirement that the internal representation in the computer memory be in this same order.

One of the guidelines used for the development of this volume of POSIX.1-2024 is that characters outside the invariant part of the ISO/IEC 646:1991 standard should not be used in portable specifications. The <backslash> character is not in the invariant part; the <number-sign> is, but with multiple representations: as a <number-sign>, and as a hash sign. As far as general usage of these symbols, they are covered by the ``grandfather clause'', but for newly defined interfaces, the WG15 POSIX working group has requested that POSIX provide alternate representations. Consequently, while the default escape character remains the <backslash> and the default comment character is the <number-sign>, implementations are required to recognize alternative representations, identified in the applicable source file via the **<escape_char>** and **<comment_char>** keywords.

A.7.3.1 *LC_CTYPE*

The *LC_CTYPE* category is primarily used to define the encoding-independent aspects of a character set, such as character classification. In addition, certain encoding-dependent characteristics are also defined for an application via the *LC_CTYPE* category. POSIX.1-2024 does not mandate that the encoding used in the locale is the same as the one used by the application because an implementation may decide that it is advantageous to define locales in a system-wide encoding rather than having multiple, logically identical locales in different encodings, and to convert from the application encoding to the system-wide encoding on usage. Other implementations could require encoding-dependent locales.

In either case, the *LC_CTYPE* attributes that are directly dependent on the encoding, such as **<mb_cur_max>** and the display width of characters, are not user-specifiable in a locale source and are consequently not defined as keywords.

Implementations may define additional keywords or extend the *LC_CTYPE* mechanism to allow application-defined keywords.

The text ``The ellipsis specification shall only be valid within a single encoded character set'' is present because it is possible to have a locale supported by multiple character encodings, as explained in the rationale for XBD Section 6.1 (on page 117). An example given there is of a possible Japanese-based locale supported by a mixture of the character sets JIS X 0201 Roman, JIS X 0208, and JIS X 0201 Katakana. Attempting to express a range of characters across these sets is not logical and the implementation is free to reject such attempts.

As the *LC_CTYPE* character classes are based on the ISO C standard character class definition, the category does not support multi-character elements. For instance, the German character <sharp-s> is traditionally classified as a lowercase letter. There is no corresponding uppercase letter; in proper capitalization of German text, the <sharp-s> will be replaced by `"SS"`; that is, by two characters. This kind of conversion is outside the scope of the **toupper** and **tolower** keywords.

Where POSIX.1-2024 specifies that only certain characters can be specified, as for the keywords **digit** and **xdigit**, the specified characters must be from the portable character set, as shown. As

126687 an example, only the Arabic digits 0 through 9 are acceptable as digits.

126688 The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included
126689 characters. These only need to be specified if the character values (that is, encoding) differs from
126690 the implementation default values. It is not possible to define a locale without these
126691 automatically included characters unless some implementation extension is used to prevent
126692 their inclusion. Such a definition would not be a proper superset of the C locale, and thus, it
126693 might not be possible for the standard utilities to be implemented as programs conforming to
126694 the ISO C standard.

126695 The definition of character class **digit** requires that only ten characters—the ones defining
126696 digits—can be specified; alternate digits (for example, Hindi or Kanji) cannot be specified here.
126697 However, the encoding may vary if an implementation supports more than one encoding.

126698 The definition of character class **xdigit** requires that the characters included in character class
126699 **digit** are included here also and allows for different symbols for the hexadecimal digits 10
126700 through 15.

126701 The inclusion of the **charclass** keyword satisfies the following requirement from the
126702 ISO POSIX-2: 1993 standard, Annex H.1:

126703 (3) *The LC_CTYPE (2.5.2.1) locale definition should be enhanced to allow user-specified additional*
126704 *character classes, similar in concept to the ISO C standard Multibyte Support Extension (MSE)*
126705 *iswctype( ) function.*

126706 This keyword was previously included in The Open Group specifications and is now mandated
126707 in the Shell and Utilities volume of POSIX.1-2024.

126708 The symbolic constant {CHARCLASS_NAME_MAX} was also adopted from The Open Group
126709 specifications. Applications portability is enhanced by the use of symbolic constants.

126710 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0033 [663], XBD/TC2-2008/0034 [663],
126711 XBD/TC2-2008/0035 [584], and XBD/TC2-2008/0036 [584] are applied.

126712 Austin Group Defect 1078 is applied, clarifying that only the specified set of characters can be
126713 included in the **digit** and **xdigit** classes in all locales, not just the POSIX locale.

126714 Austin Group Defect 1589 is applied, disallowing some characters from being included in the
126715 **blank** class.

126716 ### *A.7.3.2 LC_COLLATE*

126717 The rules governing collation depend to some extent on the use. At least five different levels of
126718 increasingly complex collation rules can be distinguished:

126719 1. *Byte/machine code order*: This is the historical collation order in the UNIX system and many
126720 proprietary operating systems. Collation is here performed character by character,
126721 without any regard to context. The primary virtue is that it usually is quite fast and also
126722 completely deterministic; it works well when the native machine collation sequence
126723 matches the user expectations.

126724 2. *Character order*: On this level, collation is also performed character by character, without
126725 regard to context. The order between characters is, however, not determined by the code
126726 values, but on the expectations by the user of the ``correct'' order between characters. In
126727 addition, such a (simple) collation order can specify that certain characters collate equally
126728 (for example, uppercase and lowercase letters).

3. *String ordering*: On this level, entire strings are compared based on relatively straightforward rules. Several ``passes'' may be required to determine the order between two strings. Characters may be ignored in some passes, but not in others; the strings may be compared in different directions; and simple string substitutions may be performed before strings are compared. This level is best described as ``dictionary'' ordering; it is based on the spelling, not the pronunciation, or meaning, of the words.

4. *Text search ordering*: This is a further refinement of the previous level, best described as ``telephone book ordering''; some common homonyms (words spelled differently but with the same pronunciation) are collated together; numbers are collated as if they were spelled out, and so on.

5. *Semantic-level ordering*: Words and strings are collated based on their meaning; entire words (such as ``the'') are eliminated; the ordering is not deterministic. This usually requires special software and is highly dependent on the intended use.

While the historical collation order formally is at level 1, for the English language it corresponds roughly to elements at level 2. The user expects to see the output from the *ls* utility sorted very much as it would be in a dictionary. While telephone book ordering would be an optimal goal for standard collation, this was ruled out as the order would be language-dependent. Furthermore, a requirement was that the order must be determined solely from the text string and the collation rules; no external information (for example, ``pronunciation dictionaries'') could be required.

As a result, the goal for the collation support is at level 3. This also matches the requirements for the Canadian collation order, as well as other, known collation requirements for alphabetic scripts. It specifically rules out collation based on pronunciation rules or based on semantic analysis of the text.

The syntax for the *LC_COLLATE* category source meets the requirements for level 3 and has been verified to produce the correct result with examples based on French, Canadian, and Danish collation order. Because it supports multi-character collating elements, it is also capable of supporting collation in codesets where a character is expressed using non-spacing characters followed by the base character (such as the ISO/IEC 6937: 2001 standard).

The directives that can be specified in an operand to the **order_start** keyword are based on the requirements specified in several proposed standards and in customary use. The following is a rephrasing of rules defined for ``lexical ordering in English and French'' by the Canadian Standards Association (the text in square brackets is rephrased):

- Once special characters [punctuation] have been removed from original strings, the ordering is determined by scanning forwards (left to right) [disregarding case and diacriticals].

- In case of equivalence, special characters are once again removed from original strings and the ordering is determined by scanning backwards (starting from the rightmost character of the string and back), character by character [disregarding case but considering diacriticals].

- In case of repeated equivalence, special characters are removed again from original strings and the ordering is determined by scanning forwards, character by character [considering both case and diacriticals].

- If there is still an ordering equivalence after the first three rules have been applied, then only special characters and the position they occupy in the string are considered to determine ordering. The string that has a special character in the lowest position comes first. If two strings have a special character in the same position, the character [with the lowest collation value] comes first. In case of equality, the other special characters are

126777 considered until there is a difference or until all special characters have been exhausted.

126778 It is estimated that this part of POSIX.1-2024 covers the requirements for all European
126779 languages, and no particular problems are anticipated with Slavic or Middle East character sets.

126780 The Far East (particularly Japanese/Chinese) collations are often based on contextual
126781 information and pronunciation rules (the same ideogram can have different meanings and
126782 different pronunciations). Such collation, in general, falls outside the desired goal of
126783 POSIX.1-2024. There are, however, several other collation rules (stroke/radical or ``most
126784 common pronunciation'') that can be supported with the mechanism described here.

126785 The character order is defined by the order in which characters and elements are specified
126786 between the **order_start** and **order_end** keywords. Weights assigned to the characters and
126787 elements define the collation sequence; in the absence of weights, the character order is also the
126788 collation sequence.

126789 The **position** keyword provides the capability to consider, in a compare, the relative position of
126790 characters not subject to **IGNORE**. As an example, consider the two strings `"o-ring"` and
126791 `"or-ing"`. Assuming the <hyphen-minus> is subject to **IGNORE** on the first pass, the two
126792 strings compare equal, and the position of the <hyphen-minus> is immaterial. On second pass,
126793 all characters except the <hyphen-minus> are subject to **IGNORE**, and in the normal case the
126794 two strings would again compare equal. By taking position into account, the first collates before
126795 the second.

126796 This standard requires that all implementation-provided locales define a collation sequence that
126797 has a total ordering of all characters unless the locale name has an `'@'` modifier indicating that
126798 it has a special collation sequence. Defining locales in this way eliminates unexpected behavior
126799 when non-identical strings can collate equally (for example, `sort -u` and `sort | uniq` are
126800 not equivalent). The exception for locales with a suitable `'@'` modifier in the name allows
126801 implementations to supply locales which do not have a total ordering of all characters provided
126802 that they draw attention to it in the modifier name. For example, `@icase` could indicate that
126803 each upper and lowercase character pair collates equally. Even with an `'@'` modifier, total
126804 ordering is preferred when possible; for example, characters that are ``ignored'' in dictionary
126805 order need not be completely ignored (by using **IGNORE** for all collation weights), but can
126806 instead be given a unique weight after one or more **IGNORE** weights.

126807 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0037 [938], XBD/TC2-2008/0038 [663],
126808 and XBD/TC2-2008/0039 [584] are applied.

126809 Austin Group Defect 948 is applied, requiring that all implementation-provided locales define a
126810 collation sequence that has a total ordering of all characters unless the locale name has an `'@'`
126811 modifier indicating that it has a special collation sequence.

126812 Austin Group Defect 1740 is applied, noting that it is the responsibility of the locale writer to
126813 ensure <NUL> has the lowest primary weight in a collation ordering.

126814 *A.7.3.3 LC_MONETARY*

126815 The currency symbol does not appear in *LC_MONETARY* because it is not defined in the C
126816 locale of the ISO C standard.

126817 The ISO C standard limits the size of decimal points and thousands delimiters to single-byte
126818 values. In locales based on multi-byte coded character sets, this cannot be enforced;
126819 POSIX.1-2024 does not prohibit such characters, but makes the behavior unspecified (in the text
126820 ``In contexts where other standards ...'').

126821 The grouping specification is based on, but not identical to, the ISO C standard. The –1 indicates
126822 that no further grouping is performed; the equivalent of {CHAR_MAX} in the ISO C standard.

The text ``the value is not available in the locale'' is taken from the ISO C standard and is used instead of the ``unspecified'' text in early proposals. There is no implication that omitting these keywords or assigning them values of `""` or –1 produces unspecified results; such omissions or assignments eliminate the effects described for the keyword or produce zero-length strings, as appropriate.

The locale definition is an extension of the ISO C standard *localeconv*( ) specification. In particular, rules on how **currency_symbol** is treated are extended to also cover **int_curr_symbol**, and **p_set_by_space** and **n_sep_by_space** have been augmented with the value 2, which places a <space> between the sign and the symbol. This has been updated to match the ISO/IEC 9899:1999 standard requirements and is an incompatible change from UNIX 98 and the ISO POSIX-2 standard and the ISO POSIX-1:1996 standard requirements. The following table shows the result of various combinations:

		p_sep_by_space		
		2	**1**	**0**
p_cs_precedes = 1	**p_sign_posn** = 0	`($1.25)`	`($ 1.25)`	`($1.25)`
	p_sign_posn = 1	`+ $1.25`	`+$ 1.25`	`+$1.25`
	p_sign_posn = 2	`$1.25 +`	`$ 1.25+`	`$1.25+`
	p_sign_posn = 3	`+ $1.25`	`+$ 1.25`	`+$1.25`
	p_sign_posn = 4	`$ +1.25`	`$+ 1.25`	`$+1.25`
p_cs_precedes = 0	**p_sign_posn** = 0	`(1.25 $)`	`(1.25 $)`	`(1.25$)`
	p_sign_posn = 1	`+1.25 $`	`+1.25 $`	`+1.25$`
	p_sign_posn = 2	`1.25$ +`	`1.25 $+`	`1.25$+`
	p_sign_posn = 3	`1.25+ $`	`1.25 +$`	`1.25+$`
	p_sign_posn = 4	`1.25$ +`	`1.25 $+`	`1.25$+`

The following is an example of the interpretation of the **mon_grouping** keyword. Assuming that the value to be formatted is 123 456 789 and the **mon_thousands_sep** is <apostrophe>, then the following table shows the result. The third column shows the equivalent string in the ISO C standard that would be used by the *localeconv*( ) function to accommodate this grouping.

mon_grouping	**Formatted Value**	**ISO C String**
3;–1	123456'789	`"\3\177"`
3	123'456'789	`"\3"`
3;2;–1	1234'56'789	`"\3\2\177"`
3;2	12'34'56'789	`"\3\2"`
–1	123456789	`"\177"`

In these examples, the octal value of {CHAR_MAX} is 177.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/6 adds a correction that permits the Euro currency symbol and addresses extensibility. The correction is stated using the term ``should'' intentionally, in order to make this a recommendation rather than a restriction on implementations. This allows for flexibility in implementations on how they handle future currency symbol additions.

IEEE Std 1003.1-2001/Cor 1-2002, tem XBD/TC1/D6/5 is applied, adding the **int_[np]_*** values to the POSIX locale definition of *LC_MONETARY*.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/16 is applied, updating the descriptions of **p_sep_by_space**, **n_sep_by_space**, **int_p_sep_by_space**, and **int_n_sep_by_space** to match the description of these keywords in the ISO C standard and the System Interfaces volume of POSIX.1-2024, *localeconv*( ).

Austin Group Defect 1199 is applied, adding a requirement that *localedef* does not accept certain

126870 combinations of ***_sign_posn**, **positive_sign**, and **negative_sign** values.

126871 Austin Group Defect 1241 is applied, clarifying the meaning of empty string values.

126872 *A.7.3.4 LC_NUMERIC*

126873 See the rationale for *LC_MONETARY* for a description of the behavior of grouping.

126874 Austin Group Defect 1241 is applied, clarifying the meaning of empty string values.

126875 *A.7.3.5 LC_TIME*

126876 Although certain of the conversion specifications in the POSIX locale (such as the name of the
126877 month) are shown with initial capital letters, this need not be the case in other locales. Programs
126878 using these conversion specifications may need to adjust the capitalization if the output is going
126879 to be used at the beginning of a sentence.

126880 The *LC_TIME* descriptions of **abday**, **day**, **mon**, and **abmon** imply a Gregorian style calendar
126881 (7-day weeks, 12-month years, leap years, and so on). Formatting time strings for other types of
126882 calendars is outside the scope of POSIX.1-2024.

126883 While the ISO 8601-1:2019 standard numbers the weekdays starting with Monday, historical
126884 practice is to use the Sunday as the first day. Rather than change the order and introduce
126885 potential confusion, the days must be specified beginning with Sunday; previous references to
126886 ``first day'' have been removed. Note also that the Shell and Utilities volume of POSIX.1-2024
126887 *date* utility supports numbering compliant with the ISO 8601-1:2019 standard.

126888 As specified under *date* in the Shell and Utilities volume of POSIX.1-2024 and *strftime*( ) in the
126889 System Interfaces volume of POSIX.1-2024, the conversion specifications corresponding to the
126890 optional keywords consist of a modifier followed by a traditional conversion specification (for
126891 instance, `%Ex`). If the optional keywords are not supported by the implementation or are
126892 unspecified for the current locale, these modified conversion specifications are treated as the
126893 traditional conversion specifications. For example, assume the following keywords:

```
126894 alt_digits   "0th";"1st";"2nd";"3rd";"4th";"5th";\
126895               "6th";"7th";"8th";"9th";"10th"

126896 d_fmt        "The %Od day of %B in %Y"
```

126897 On July 4th 1776, the `%x` conversion specifications would result in `"The 4th day of July
126898 in 1776"`, while on July 14th 1789 it would result in `"The 14 day of July in 1789"`. It
126899 can be noted that the above example is for illustrative purposes only; the `%O` modifier is
126900 primarily intended to provide for Kanji or Hindi digits in *date* formats.

126901 The following is an example for Japan that supports the current plus last three Emperors and
126902 reverts to Western style numbering for years prior to the Meiji era. The example also allows for
126903 the custom of using a special name for the first year of an era instead of using 1. (The examples
126904 substitute romaji where kanji should be used.)

```
126905 era_d_fmt "%EY%mgatsu%dnichi (%a)"

126906 era    "+:2:1990/01/01:+*:Heisei:%EC%Eynen";\
126907        "+:1:1989/01/08:1989/12/31:Heisei:%ECgannen";\
126908        "+:2:1927/01/01:1989/01/07:Shouwa:%EC%Eynen";\
126909        "+:1:1926/12/25:1926/12/31:Shouwa:%ECgannen";\
126910        "+:2:1913/01/01:1926/12/24:Taishou:%EC%Eynen";\
126911        "+:1:1912/07/30:1912/12/31:Taishou:%ECgannen";\
126912        "+:2:1869/01/01:1912/07/29:Meiji:%EC%Eynen";\
```



```
126913          "+:1:1868/09/08:1868/12/31:Meiji:%ECgannen";\
126914          "-:1868:1868/09/07:-*::%Ey"
```

126915 Assuming that the current date is September 21, 1991, a request to *date* or *strftime*() would yield
126916 the following results:

```
126917 %Ec - Heisei3nen9gatsu21nichi (Sat) 14:39:26
126918 %EC - Heisei
126919 %Ex - Heisei3nen9gatsu21nichi (Sat)
126920 %Ey - 3
126921 %EY - Heisei3nen
```

126922 Example era definitions for the Republic of China:

```
126923 era     "+:2:1913/01/01:+*:ChungHwaMingGuo:%EC%EyNen";\
126924          "+:1:1912/1/1:1912/12/31:ChungHwaMingGuo:%ECYuenNen";\
126925          "+:1:1911/12/31:-*:MingChien:%EC%EyNen"
```

126926 Example definitions for the Christian Era:

```
126927 era     "+:1:0001/01/01:+*:AD:%EC %Ey";\
126928          "+:1:-0001/12/31:-*:BC:%Ey %EC"
```

126929 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0040 [912] is applied.

126930 Austin Group Defects 258 and 1166 are applied, adding the **alt_mon** and **ab_alt_mon** locale
126931 keywords.

126932 Austin Group Defect 1307 is applied, changing the **am_pm** and **t_fmt_ampm** keywords and the
126933 AM_STR, PM_STR, and T_FMT_AMPM constants in relation to locales that do not support the
126934 12-hour clock format.

126935 *A.7.3.6 LC_MESSAGES*

126936 The **yesstr** and **nostr** locale keywords and the YESSTR and NOSTR *langinfo* items were formerly
126937 used to match user affirmative and negative responses. In POSIX.1-2024, the **yesexpr**, **noexpr**,
126938 YESEXPR, and NOEXPR extended regular expressions have replaced them. Applications
126939 should use the general locale-based messaging facilities to issue prompting messages which
126940 include sample desired responses.

126941 Affirmative responses like:

```
126942 y
126943 Yes
126944 Yes!
```

126945 and negative responses like:

```
126946 N
126947 No
126948 Never
126949 No way!
```

126950 should all be recognized as affirmative and negative responses, respectively, by the EREs
126951 identified by the **yesexpr** and **noexpr** keywords for English language-based locales. There is no
126952 requirement that multi-line responses nor ambiguous responses like:

```
126953 no or yes
126954 yes or no
126955 maybe
```


be correctly classified by either of these EREs. Application writers are encouraged to include locale-specific suggestions for affirmative and negative responses in prompts.

A.7.4 Locale Definition Grammar

There is no additional rationale provided for this section.

A.7.4.1 Locale Lexical Conventions

There is no additional rationale provided for this section.

A.7.4.2 Locale Grammar

Austin Group Defects 258 and 1166 are applied, adding the **alt_mon** and **ab_alt_mon** locale keywords.

A.7.5 Locale Definition Example

The following is an example of a locale definition file that could be used as input to the *localedef* utility. It assumes that the utility is executed with the **–f** option, naming a charmap file with (at least) the following content:

```
CHARMAP
<space>      \x20
<dollar>     \x24
<A>          \101
<a>          \141
<A-acute>    \346
<a-acute>    \365
<A-grave>    \300
<a-grave>    \366
<b>          \142
<C>          \103
<c>          \143
<c-cedilla>  \347
<d>          \x64
<H>          \110
<h>          \150
<eszet>      \xb7
<s>          \x73
<z>          \x7a
END CHARMAP
```

It should not be taken as complete or to represent any actual locale, but only to illustrate the syntax.

```
#
LC_CTYPE
lower   <a>;<b>;<c>;<c-cedilla>;<d>;...;<z>
upper   A;B;C;Ç;...;Z
space   \x20;\x09;\x0a;\x0b;\x0c;\x0d
blank   \040;\011
```



```
126997        toupper (<a>,<A>);(b,B);(c,C);(ç,Ç);(d,D);(z,Z)
126998        END LC_CTYPE
126999        #
127000        LC_COLLATE
127001        #
127002        # The following example of collation is based on
127003        # Canadian standard Z243.4.1-1998, "Canadian Alphanumeric
127004        # Ordering Standard for Character Sets of CSA Z234.4 Standard".
127005        # (Other parts of this example locale definition file do not
127006        # purport to relate to Canada, or to any other real culture.)
127007        # The proposed standard defines a 4-weight collation, such that
127008        # in the first pass, characters are compared without regard to
127009        # case or accents; in the second pass, backwards-compare without
127010        # regard to case; in the third pass, forwards-compare without
127011        # regard to diacriticals. In the 3 first passes, non-alphabetic
127012        # characters are ignored; in the fourth pass, only special
127013        # characters are considered, such that "The string that has a
127014        # special character in the lowest position comes first. If two
127015        # strings have a special character in the same position, the
127016        # collation value of the special character determines ordering.
127017        #
127018        # Only a subset of the character set is used here; mostly to
127019        # illustrate the set-up.
127020        #
127021        collating-symbol <NULL>
127022        collating-symbol <LOW_VALUE>
127023        collating-symbol <LOWER-CASE>
127024        collating-symbol <SUBSCRIPT-LOWER>
127025        collating-symbol <SUPERSCRIPT-LOWER>
127026        collating-symbol <UPPER-CASE>
127027        collating-symbol <NO-ACCENT>
127028        collating-symbol <PECULIAR>
127029        collating-symbol <LIGATURE>
127030        collating-symbol <ACUTE>
127031        collating-symbol <GRAVE>
127032        # Further collating-symbols follow.
127033        #
127034        # Properly, the standard does not include any multi-character
127035        # collating elements; the one below is added for completeness.
127036        #
127037        collating_element <ch> from "<c><h>"
127038        collating_element <CH> from "<C><H>"
127039        collating_element <Ch> from "<C><h>"
127040        #
127041        order_start forward;backward;forward;forward,position
127042        #
127043        # Collating symbols are specified first in the sequence to allocate
127044        # basic collation values to them, lower than that of any character.
127045        <NULL>
127046        <LOW_VALUE>
127047        <LOWER-CASE>
127048        <SUBSCRIPT-LOWER>
127049        <SUPERSCRIPT-LOWER>
```



```
127050        <UPPER-CASE>
127051        <NO-ACCENT>
127052        <PECULIAR>
127053        <LIGATURE>
127054        <ACUTE>
127055        <GRAVE>
127056        <RING-ABOVE>
127057        <DIAERESIS>
127058        <TILDE>
127059        # Further collating symbols are given a basic collating value here.
127060        #
127061        # Here follow special characters.
127062        <space>        IGNORE;IGNORE;IGNORE;<space>
127063        # Other special characters follow here.
127064        #
127065        # Here follow the regular characters.
127066        <a>        <a>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
127067        <A>        <a>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
127068        <a-acute>  <a>;<ACUTE>;<LOWER-CASE>;IGNORE
127069        <A-acute>  <a>;<ACUTE>;<UPPER-CASE>;IGNORE
127070        <a-grave>  <a>;<GRAVE>;<LOWER-CASE>;IGNORE
127071        <A-grave>  <a>;<GRAVE>;<UPPER-CASE>;IGNORE
127072        <ae>      "<a><e>";"<LIGATURE><LIGATURE>";\
127073                  "<LOWER-CASE><LOWER-CASE>";IGNORE
127074        <AE>      "<a><e>";"<LIGATURE><LIGATURE>";\
127075                  "<UPPER-CASE><UPPER-CASE>";IGNORE
127076        <b>        <b>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
127077        <B>        <b>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
127078        <c>        <c>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
127079        <C>        <c>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
127080        <ch>       <ch>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
127081        <Ch>       <ch>;<NO-ACCENT>;<PECULIAR>;IGNORE
127082        <CH>       <ch>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
127083        #
127084        # As an example, the strings "Bach" and "bach" could be encoded (for
127085        # compare purposes) as:
127086        # "Bach"  <b>;<a>;<ch>;<LOW_VALUE>;<NO_ACCENT>;<NO_ACCENT>;\
127087        #         <NO_ACCENT>;<LOW_VALUE>;<UPPER-CASE>;<LOWER-CASE>;\
127088        #         <LOWER-CASE>;<NULL>
127089        # "bach"  <b>;<a>;<ch>;<LOW_VALUE>;<NO_ACCENT>;<NO_ACCENT>;\
127090        #         <NO_ACCENT>;<LOW_VALUE>;<LOWER-CASE>;<LOWER-CASE>;\
127091        #         <LOWER-CASE>;<NULL>
127092        #
127093        # The two strings are equal in pass 1 and 2, but differ in pass 3.
127094        #
127095        # Further characters follow.
127096        #
127097        UNDEFINED    IGNORE;IGNORE;IGNORE;IGNORE
127098        #
127099        order_end
127100        #
127101        END LC_COLLATE
127102        #
```



```
        LC_MONETARY
        int_curr_symbol    "USD "
        currency_symbol    "$"
        mon_decimal_point  "."
        mon_grouping       3;0
        positive_sign      ""
        negative_sign      "-"
        p_cs_precedes      1
        n_sign_posn        0
        END LC_MONETARY
        #
        LC_NUMERIC
        copy "US_en.ASCII"
        END LC_NUMERIC
        #
        LC_TIME
        abday   "Sun";"Mon";"Tue";"Wed";"Thu";"Fri";"Sat"
        #
        day     "Sunday";"Monday";"Tuesday";"Wednesday";\
                "Thursday";"Friday";"Saturday"
        #
        abmon   "Jan";"Feb";"Mar";"Apr";"May";"Jun";\
                 "Jul";"Aug";"Sep";"Oct";"Nov";"Dec"
        #
        mon     "January";"February";"March";"April";\
                "May";"June";"July";"August";"September";\
                "October";"November";"December"
        #
        d_t_fmt "%a %b %d %T %Z %Y\n"
        END LC_TIME
        #
        LC_MESSAGES
        yesexpr "^([yY][[:alpha:]]*)|(OK)"
        #
        noexpr  "^[nN][[:alpha:]]*"
        END LC_MESSAGES
```

A.8 Environment Variables

A.8.1 Environment Variable Definition

The variable *environ* is not intended to be declared in any header, but rather to be declared by the user for accessing the array of strings that is the environment. This is the traditional usage of the symbol. Putting it into a header could break some programs that use the symbol for their own purposes.

The decision to restrict conforming systems to the use of digits, uppercase letters, and underscores for environment variable names allows applications to use lowercase letters in their environment variable names without conflicting with any conforming system.

In addition to the obvious conflict with the shell syntax for positional parameter substitution, some historical applications (including some shells) exclude names with leading digits from the environment.

Some historical implementations removed certain environment variables during program startup when security criteria were not met, instead of just ignoring them at the point of use. The standard developers decided not to allow this behavior because if a process drops all privileges and sets its effective user and group IDs to be the same as its real user and group IDs before executing a program or utility, the behavior should be the same as if the process had originally met the security criteria.

Austin Group Defect 367 is applied, adding requirements relating to the use of *readonly* on environment variables that are manipulated by shell built-in utilities.

Austin Group Defect 922 is applied, allowing implementations to ignore some environment variables at the point of use for security reasons.

Austin Group Defect 1561 is applied, clarifying that environment variable values can contain byte sequences that do not form valid characters.

A.8.2 Internationalization Variables

Utilities conforming to the Shell and Utilities volume of POSIX.1-2024 and written in standard C can access the locale variables by issuing the following call:

```
setlocale(LC_ALL, "")
```

If this were omitted, the ISO C standard specifies that the C (or POSIX) locale would be used.

The DESCRIPTION of *setlocale*( ) requires that when setting all categories of a locale, if the value of any of the environment variable searches yields a locale that is not supported (and non-null), the *setlocale*( ) function returns a null pointer and the global locale is unchanged.

For the standard utilities, if any of the environment variables are invalid, it makes sense to default to an implementation-defined, consistent locale environment. It is more confusing for a user to have partial settings occur in case of a mistake. All utilities would then behave in one language/cultural environment. Furthermore, it provides a way of forcing the whole environment to be the implementation-defined default. Disastrous results could occur if a pipeline of utilities partially uses the environment variables in different ways. In this case, it would be appropriate for utilities that use *LANG* and related variables to exit with an error if any of the variables are invalid. For example, users typing individual commands at a terminal might want *date* to work if *LC_MONETARY* is invalid as long as *LC_TIME* is valid. Since these are conflicting reasonable alternatives, POSIX.1-2024 leaves the results unspecified if the locale environment variables would not produce a complete locale matching the specification of the user.

The *LC_MESSAGES* variable affects the language of messages generated by the standard utilities.

The description of the environment variable names starting with the characters ``LC_'' acknowledges the fact that the interfaces presented may be extended as new international functionality is required. In the ISO C standard, names preceded by ``LC_'' are reserved in the name space for future categories.

To avoid name clashes, new categories and environment variables are divided into two classifications: ``implementation-independent'' and ``implementation-defined''.

Implementation-independent names will have the following format:

127192 `LC_`*`NAME`*

127193 where *NAME* is the name of the new category and environment variable. Capital letters must be
127194 used for implementation-independent names.

127195 Implementation-defined names must be in lowercase letters, as below:

127196 `LC_`*`name`*

127197 Austin Group Defect 1122 is applied, adding the *LANGUAGE*, *TEXTDOMAIN*, and
127198 *TEXTDOMAINDIR* environment variables and updating *NLSPATH* with requirements relating
127199 to the *gettext* family of functions and the *gettext* and *ngettext* utilities.

127200 Austin Group Defect 1477 is applied, moving a paragraph of rationale about incompatible locale
127201 categories to Section A.7.1 (on page 3692).

127202 Austin Group Defect 1571 is applied, simplifying the final item in the precedence order for
127203 internationalization environment variables.

127204 A.8.3 Other Environment Variables

127205 **COLUMNS, LINES**

127206 The default values for the number of column positions when *COLUMNS* is unset or null, and
127207 screen height when *LINES* is unset or null, are unspecified if the terminal window size cannot be
127208 obtained (from *tcgetwinsize*()) because historical implementations use different methods to
127209 determine the values. Users should not need to set these variables in the environment unless
127210 there is a specific reason to override the default behavior of the implementation, such as to
127211 display data in an area arbitrarily smaller than the terminal or window. Values for these
127212 variables that are not decimal integers greater than zero are implicitly undefined values; it is
127213 unnecessary to enumerate all of the possible values outside of the acceptable set.

127214 Austin Group Defect 1185 is applied, changing the descriptions of the *COLUMNS* and *LINES*
127215 environment variables.

127216 **LOGNAME**

127217 In most implementations, the value of such a variable is easily forged, so security-critical
127218 applications should rely on other means of determining user identity. *LOGNAME* is required to
127219 be constructed from the portable filename character set for reasons of interchange. No diagnostic
127220 condition is specified for violating this rule, and no requirement for enforcement exists. The
127221 intent of the requirement is that if extended characters are used, the ``guarantee'' of portability
127222 implied by a standard is void.

127223 **PATH**

127224 Many historical implementations of the Bourne shell do not interpret a trailing <colon> to
127225 represent the current working directory and are thus non-conforming. The C Shell and the
127226 KornShell conform to POSIX.1-2024 on this point. The usual name of dot may also be used to
127227 refer to the current working directory.

127228 Many implementations historically have used a default value of **/bin** and **/usr/bin** for the *PATH*
127229 variable. POSIX.1-2024 does not mandate this default path be identical to that retrieved from
127230 *getconf PATH* because it is likely that the standardized utilities may be provided in another
127231 directory separate from the directories used by some historical applications.

127232 The standard specifies that (when no <slash> character is included in a command pathname)
127233 special built-in utilities and intrinsic utilities are not subject to a search using *PATH*. All other

127234 standard utilities, even if implemented as shell built-ins, are required to be found by searching
127235 *PATH*. This means that if a shell includes a built-in for a standard utility that is not intrinsic, a
127236 user can write a utility that will override that built-in. The standard also requires that all
127237 standard utilities can be executed by commands like:

127238 ```
find . -type d -exec printf 'Found directory: %s\n' '{}' +
```

127239 So, other than differences caused by using different shell execution environments, a standard
127240 utility that is implemented as a built-in and the non-built-in version of that standard utility are
127241 both required to behave as the standard specifies. But, if a non-standard utility is found in *PATH*
127242 before the standard utility's location in *PATH*, the non-standard utility must be invoked rather
127243 than the built-in. For instance, if the shell includes a built-in *printf* utility (which most shells do),
127244 *PATH* is initialized using:

127245 ```
PATH="$HOME/bin:$(command -p getconf PATH)"
```

127246 and **$HOME/bin/printf** is an executable file containing:

127247 ```
command -p printf 'In %s with args:\n' "${0##*/}" >&2
127248 command -p printf '%s\n' "$@" >&2
127249 command -V printf >&2
127250 command -Vp printf >&2
127251 command -p printf "$@"
```

127252 then the command:

127253 ```
printf '%s %s\n' HOME "$HOME" PATH "$PATH"
```

127254 should produce output similar to:

127255 ```
In printf with args:
127256   %s %s\n
127257   HOME
127258   /Users/dwc
127259   PATH
127260   /Users/dwc/bin:/usr/bin:/bin:/usr/sbin:/sbin
127261 printf is a tracked alias for /Users/dwc/bin/printf
127262 printf is a shell builtin
127263 HOME /Users/dwc
127264 PATH /Users/dwc/bin:/usr/bin:/bin:/usr/sbin:/sbin
```

127265 The current version of the Korn shell installs built-ins into the shell using a *builtin* utility that
127266 allows the built-in to be associated with the pathname of the non-built-in version of that utility.
127267 (Unfortunately, some implementations that use *ksh93* as their standard *sh* utility do not make use
127268 of this feature and install built-ins for standard utilities that are not associated with a *PATH*
127269 search. And, most other shells incorrectly always use a built-in utility if one is installed, even
127270 when it should be overridden by a *PATH* search that should find the non-standard version of a
127271 utility with the name of that built-in.) Some other shells use a <percent-sign> character in a
127272 directory pathname in *PATH* to indicate one or more directories that should be used when
127273 processing *PATH* to determine when non-intrinsic standard utilities should be found. The
127274 POSIX.1-2024 revision of the standard allows either of these methods to be used to install built-
127275 ins that meet the requirements stated in XCU Section 2.9.1.4 (on page 2502) by making the
127276 behavior of the built-in path search implementation-defined when a <percent-sign> character is
127277 found in *PATH*.

127278 Austin Group Defect 854 is applied, changing how *PATH* searching applies to built-in utilities.

127279 Austin Group Defect 1340 is applied, clarifying the description of *PATH*.

  
```

SHELL

The *SHELL* variable names the preferred shell of the user; it is a guide to applications. There is no direct requirement that that shell conform to POSIX.1-2024; that decision should rest with the user. It is the intention of the standard developers that alternative shells be permitted, if the user chooses to develop or acquire one. An operating system that builds its shell into the ``kernel'' in such a manner that alternative shells would be impossible does not conform to the spirit of POSIX.1-2024.

TZ

The quoted form of the timezone variable allows timezone names of the form UTC+1 (or any name that contains the <plus-sign> (`'+'`), the <hyphen-minus> (`'−'`), or digits), which may be appropriate for countries that do not have an official timezone name. It would be coded as <UTC+1>+1<UTC+2>, which would cause *std* to have a value of UTC+1 and *dst* a value of UTC+2, each with a length of 5 characters. This does not appear to conflict with any existing usage. The characters `'<'` and `'>'` were chosen for quoting because they are easier to parse visually than a quoting character that does not provide some sense of bracketing (and in a string like this, such bracketing is helpful). They were also chosen because they do not need special treatment when assigning to the *TZ* variable. Users are often confused by embedding quotes in a string. Because `'<'` and `'>'` are meaningful to the shell, the whole string would have to be quoted, but that is easily explained. (Parentheses would have presented the same problems.) Although the `'>'` symbol could have been permitted in the string by either escaping it or doubling it, it seemed of little value to require that. This could be provided as an extension if there was a need. Timezone names of this new form lead to a requirement that the value of {_POSIX_TZNAME_MAX} change from 3 to 6.

Since the *TZ* environment variable is usually inherited by all applications started by a user after the value of the *TZ* environment variable is changed and since many applications run using the C or POSIX locale, using characters that are not in the portable character set in the *std* and *dst* fields could cause unexpected results.

Implementations are encouraged to incorporate the IANA timezone database into the timezone database used for *TZ* values specifying geographical and special timezones, and to provide a method to allow it to be updated in accordance with RFC 6557.

The *TZ* format beginning with <colon> was originally introduced as a way for implementations to support geographical timezones in the form `:Area/Location` as an extension, but implementations started to support them without the leading <colon> (as well as with it) and their use without the <colon> became the de-facto standard. Consequently when geographical timezones were added to this standard, it was without the <colon>.

The format of the *TZ* environment variable is changed in Issue 6 to allow for the quoted form, as defined in earlier versions of the ISO POSIX-1 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/7 is applied, adding the *ctime_r*( ) and *localtime_r*( ) functions to the list of functions that use the *TZ* environment variable.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0041 [584] is applied.

Austin Group Defect 1030 is applied, making it implementation-defined when the changes to and from Daylight Saving Time occur if the *dst* field is specified in *TZ* and the *rule* field is not.

Austin Group Defect 1252 is applied, changing the *time* field to allow the hour to range from zero to 167 and allowing a leading sign.

Austin Group Defect 1253 is applied, changing ``alternative time'' to ``Daylight Saving Time''.

Austin Group Defect 1410 is applied, removing the *ctime_r*( ) function.

Austin Group Defect 1619 is applied, adding support for a third *TZ* format with values specifying geographical and special timezones.

Austin Group Defects 1638 and 1639 are applied, clarifying the length limits for the *std* and *dst* fields of *TZ*.

A.9 Regular Expressions

Rather than repeating the description of REs for each utility supporting REs, the standard developers preferred a common, comprehensive description of regular expressions in one place. The most common behavior is described here, and exceptions or extensions to this are documented for the respective utilities, as appropriate.

The BRE corresponds to the *ed* or historical *grep* type, and the ERE corresponds to the historical *egrep* type (now *grep* **–E**).

The text is based on the *ed* description and substantially modified, primarily to aid developers and others in the understanding of the capabilities and limitations of REs. Much of this was influenced by internationalization requirements.

It should be noted that the definitions in this section do not cover the *tr* utility; the *tr* syntax does not employ REs.

The specification of REs is particularly important to internationalization because pattern matching operations are very basic operations in business and other operations. The syntax and rules of REs are intended to be as intuitive as possible to make them easy to understand and use. The historical rules and behavior do not provide that capability to non-English language users, and do not provide the necessary support for commonly used characters and language constructs. It was necessary to provide extensions to the historical RE syntax and rules to accommodate other languages.

As they are limited to bracket expressions, the rationale for these modifications is in XBD Section 9.3.5 (on page 182).

A.9.1 Regular Expression Definitions

It is possible to determine what strings correspond to subexpressions by recursively applying the leftmost longest rule to each subexpression, but only with the proviso that the overall match is leftmost longest. For example, matching `"\(ac*\)c*d[ac]*\1"` against *acdacaaa* matches *acdacaaa* (with \1=*a*); simply matching the longest match for `"\(ac*\)"` would yield \1=*ac*, but the overall match would be smaller (*acdac*). Conceptually, the implementation must examine every possible match and among those that yield the leftmost longest total matches, pick the one that does the longest match for the leftmost subexpression, and so on. Note that this means that matching by subexpressions is context-dependent: a subexpression within a larger RE may match a different string from the one it would match as an independent RE, and two instances of the same subexpression within the same larger RE may match different lengths even in similar sequences of characters. For example, in the ERE `"(a.*b)(a.*b)"`, the two identical subexpressions would match four and six characters, respectively, of *accbaccccb*.

The definition of single character has been expanded to include also collating elements consisting of two or more characters; this expansion is applicable only when a bracket expression is included in the BRE or ERE. An example of such a collating element may be the Dutch *ij*, which collates as a `'y'`. In some encodings, a ligature ``i with j'' exists as a character and would represent a single-character collating element. In another encoding, no such ligature exists, and the two-character sequence *ij* is defined as a multi-character collating element.

Outside brackets, the *ij* is treated as a two-character RE and matches the same characters in a string. Historically, a bracket expression only matched a single character. The ISO POSIX-2: 1993 standard required bracket expressions like `"[^[:lower:]]"` to match multi-character collating elements such as `"ij"`. However, this requirement led to behavior that many users did not expect and that could not feasibly be mimicked in user code, and it was rarely if ever implemented correctly. The current standard leaves it unspecified whether a bracket expression matches a multi-character collating element, allowing both historical and ISO POSIX-2: 1993 standard implementations to conform.

Also, in the current standard, it is unspecified whether character class expressions like `"[:lower:]"` can include multi-character collating elements like `"ij"`; hence `"[[:lower:]]"` can match `"ij"`, and `"[^[:lower:]]"` can fail to match `"ij"`. Common practice is for a character class expression to match a collating element if it matches the collating element's first character.

Austin Group Defect 1329 is applied, adding a definition of ``leftmost'' and updating the definition of ``matched'' to include an example ERE using the repetition modifier `'?'`.

Austin Group Defect 1546 is applied, adding a definition of ``escape sequence''.

A.9.2 Regular Expression General Requirements

The definition of which sequence is matched when several are possible is based on the leftmost-longest rule historically used by deterministic recognizers. This rule is easier to define and describe, and arguably more useful, than the first-match rule historically used by non-deterministic recognizers. It is thought that dependencies on the choice of rule are rare; carefully contrived examples are needed to demonstrate the difference.

A formal expression of the leftmost-longest rule is:

> The search is performed as if all possible suffixes of the string were tested for a prefix matching the pattern; the longest suffix containing a matching prefix is chosen, and the longest possible matching prefix of the chosen suffix is identified as the matching sequence.

EREs can optionally use a leftmost-shortest rule for repetitions (enabled via the REG_MINIMAL flag or the `'?'` repetition modifier), in which case the shortest possible matching prefix is instead identified as the matching sequence for the affected repetition(s).

Historically, most RE implementations only match lines, not strings. However, that is more an effect of the usage than of an inherent feature of REs themselves. Consequently, POSIX.1-2024 does not regard <newline> characters as special; they are ordinary characters, and both a <period> and a non-matching list can match them. Those utilities (like *grep*) that do not allow <newline> characters to match are responsible for eliminating any <newline> from strings before matching against the RE. The *regcomp*( ) function, however, can provide support for such processing without violating the rules of this section.

Some implementations of *egrep* have had very limited flexibility in handling complex EREs. POSIX.1-2024 does not attempt to define the complexity of a BRE or ERE, but does place a lower limit on it—any RE must be handled, as long as it can be expressed in 256 bytes or less. (Of course, this does not place an upper limit on the implementation.) There are historical programs using a non-deterministic-recognizer implementation that should have no difficulty with this limit. It is possible that a good approach would be to attempt to use the faster, but more limited, deterministic recognizer for simple expressions and to fall back on the non-deterministic recognizer for those expressions requiring it. Non-deterministic implementations must be careful to observe the rules on which match is chosen; the longest match, not the first match,

starting at a given character is used.

The term ``invalid'' highlights a difference between this section and some others: POSIX.1-2024 frequently avoids mandating of errors for syntax violations because they can be used by implementors to trigger extensions. However, the authors of the internationalization features of REs wanted to mandate errors for certain conditions to identify usage problems or non-portable constructs. These are identified within this rationale as appropriate. The remaining syntax violations have been left implicitly or explicitly undefined. For example, the BRE construct `"\{1,2,3\}"` does not comply with the grammar. A conforming application cannot rely on it producing an error nor matching the literal characters `"\{1,2,3\}"`.

The term ``undefined'' was used in favor of ``unspecified'' because many of the situations are considered errors on some implementations, and the standard developers considered that consistency throughout the section was preferable to mixing undefined and unspecified.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0042 [554] is applied.

Austin Group Defect 1031 is applied, replacing text relating to case insensitive comparisons with a reference to XBD Section 4.1 (on page 95).

A.9.3 Basic Regular Expressions

Austin Group Defect 1139 is applied, making minor editorial changes to several subsections of this section and changing them to require that, when not inside a bracket expression, `"\]"` matches `']'`.

A.9.3.1 BREs Matching a Single Character or Collating Element

There is no additional rationale provided for this section.

A.9.3.2 BRE Ordinary Characters

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0043 [554] is applied.

Austin Group Defect 1546 is applied, adding optional support for `"\?"`, `"\+"`, and `"\|"`.

A.9.3.3 BRE Special Characters

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0043 [554] is applied.

Austin Group Defect 1546 is applied, adding optional support for `"\?"`, `"\+"`, and `"\|"`.

A.9.3.4 Periods in BREs

There is no additional rationale provided for this section.

A.9.3.5 RE Bracket Expression

Range expressions are, historically, an integral part of REs. However, the requirements of ``natural language behavior'' and portability do conflict. In the POSIX locale, ranges must be treated according to the collating sequence and include such characters that fall within the range based on that collating sequence, regardless of character values. In other locales, ranges have unspecified behavior.

Some historical implementations allow range expressions where the ending range point of one range is also the starting point of the next (for instance, `"[a-m-o]"`). This behavior should not be permitted, but to avoid breaking historical implementations, it is now *undefined* whether it is a valid expression and how it should be interpreted.

Current practice in *awk* and *lex* is to accept escape sequences in bracket expressions as per XBD Table 5-1 (on page 113), while the normal ERE behavior is to regard such a sequence as consisting of two characters. Allowing the *awk*/*lex* behavior in EREs would change the normal behavior in an unacceptable way; it is expected that *awk* and *lex* will decode escape sequences in EREs before passing them to *regcomp*( ) or comparable routines. Each utility describes the escape sequences it accepts as an exception to the rules in this section; the list is not the same, for historical reasons.

As noted previously, the new syntax and rules have been added to accommodate other languages than English. The remainder of this section describes the rationale for these modifications.

In the POSIX locale, a regular expression that starts with a range expression matches a set of strings that are contiguously sorted, but this is not necessarily true in other locales. For example, a French locale might have the following behavior:

```
$ ls
alpha   Alpha   estimé   ESTIMÉ   été   eurêka
$ ls [a-e]*
alpha   Alpha   estimé  eurêka
```

Such disagreements between matching and contiguous sorting are unavoidable because POSIX sorting cannot be implemented in terms of a deterministic finite-state automaton (DFA), but range expressions by design are implementable in terms of DFAs.

Historical implementations used native character order to interpret range expressions. The ISO POSIX-2:1993 standard instead required collating element order (CEO): the order that collating elements were specified between the **order_start** and **order_end** keywords in the *LC_COLLATE* category of the current locale. CEO had some advantages in portability over the native character order, but it also had some disadvantages:

- CEO could not feasibly be mimicked in user code, leading to inconsistencies between POSIX matchers and matchers in popular user programs like Emacs, *ksh*, and Perl.
- CEO caused range expressions to match accented and capitalized letters contrary to many users' expectations. For example, `"[a-e]"` typically matched both `'E'` and `'á'` but neither `'A'` nor `'é'`.
- CEO was not consistent across implementations. In practice, CEO was often less portable than native character order. For example, it was common for the CEOs of two implementation-supplied locales to disagree, even if both locales were named `"da_DK"`.

Because of these problems, some implementations of regular expressions continued to use native character order. Others used the collation sequence, which is more consistent with sorting than either CEO or native order, but which departs further from the traditional POSIX semantics because it generally requires `"[a-e]"` to match either `'A'` or `'E'` but not both. As a result of this kind of implementation variation, programmers who wanted to write portable regular expressions could not rely on the ISO POSIX-2:1993 standard guarantees in practice.

While revising the standard, lengthy consideration was given to proposals to attack this problem by adding an API for querying the CEO to allow user-mode matchers, but none of these proposals had implementation experience and none achieved consensus. Leaving the standard alone was also considered, but rejected due to the problems described above.

The current standard leaves unspecified the behavior of a range expression outside the POSIX locale. This makes it clearer that conforming applications should avoid range expressions outside the POSIX locale, and it allows implementations and compatible user-mode matchers to interpret range expressions using native order, CEO, collation sequence, or other, more advanced techniques. The concerns which led to this change were raised in IEEE PASC interpretation 1003.2 #43 and others, and related to ambiguities in the specification of how multi-character collating elements should be handled in range expressions. These ambiguities had led to multiple interpretations of the specification, in conflicting ways, which led to varying implementations. As noted above, efforts were made to resolve the differences, but no solution has been found that would be specific enough to allow for portable software while not invalidating existing implementations.

The standard developers recognize that collating elements are important, such elements being common in several European languages; for example, `'ch'` or `'ll'` in traditional Spanish; `'aa'` in several Scandinavian languages. Existing internationalized implementations have processed, and continue to process, these elements in range expressions. Efforts are expected to continue in the future to find a way to define the behavior of these elements precisely and portably.

The ISO POSIX-2:1993 standard required `"[b-a]"` to be an invalid expression in the POSIX locale, but this requirement has been relaxed in this version of the standard so that `"[b-a]"` can instead be treated as a valid expression that does not match any string.

The standard specifies three possible behaviors for regular expressions such as `"[:alpha:]"`. One behavior is the traditional implementation, which behaves like `"[:ahlp]"`. Another, for alignment with the *tr* utility, is to treat it like `"[[:alpha:]]"`. And finally, the standard allows rejecting the regular expression as invalid, as a means of alerting a user to the non-portable aspect of that regular expression. The set of regular expressions with this undefined behavior is limited solely to the expressions where the outer `'['` and `']'` of the bracket expression can be confused with the missing bracket pair `'['` and `']'` necessary to form a collating symbol, equivalence class, or character class; thus `"[_:alpha:]"` or `"[::]"` do not trigger the unspecified behavior.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0044 [938], XBD/TC2-2008/0045 [872], XBD/TC2-2008/0046 [938], XBD/TC2-2008/0047 [584], and XBD/TC2-2008/0048 [584] are applied.

Austin Group Defect 948 is applied, requiring that an ordinary character in a matching list only matches that character.

Austin Group Defect 1190 is applied, clarifying which characters lose their special meaning inside a bracket expression.

Austin Group Defect 1288 is applied, changing ``rejected as an error'' to ``treated as an invalid bracket expression''.

A.9.3.6 BREs Matching Multiple Characters

The limit of nine back-references to subexpressions in the RE is based on the use of a single-digit identifier; increasing this to multiple digits would break historical applications. This does not imply that only nine subexpressions are allowed in REs. The following is a valid BRE with ten subexpressions:

```
\(\(\(ab\)*c\)*d\)\(ef\)*\(gh\)\{2\}\(ij\)*\(kl\)*\(mn\)*\(op\)*\(qr\)*
```

The standard developers regarded the common historical behavior, which supported `"\n*"`, but not `"\n\{min,max\}"`, `"\(...\)*"`, or `"\(...\)\{min,max\}"`, as a non-intentional

result of a specific implementation, and they supported both duplication and interval expressions following subexpressions and back-references.

The changes to the processing of the back-reference expression remove an unspecified or ambiguous behavior in the Shell and Utilities volume of POSIX.1-2024, aligning it with the requirements specified for the *regcomp*() expression, and is the result of PASC Interpretation 1003.2-92 #43 submitted for the ISO POSIX-2: 1993 standard.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0049 [595] is applied.

A.9.3.7 BRE Precedence

There is no additional rationale provided for this section.

A.9.3.8 BRE Expression Anchoring

Often, the <dollar-sign> is viewed as matching the ending <newline> in text files. This is not strictly true; the <newline> is typically eliminated from the strings to be matched, and the <dollar-sign> matches the terminating null character.

The ability of `'^'`, `'$'`, and `'*'` to be non-special in certain circumstances may be confusing to some programmers, but this situation was changed only in a minor way from historical practice to avoid breaking many historical scripts. Some consideration was given to making the use of the anchoring characters undefined if not escaped and not at the beginning or end of strings. This would cause a number of historical BREs, such as `"2^10"`, `"$HOME"`, and `"$1.35"`, that relied on the characters being treated literally, to become invalid.

However, one relatively uncommon case was changed to allow an extension used on some implementations. Historically, the BREs `"^foo"` and `"\(^foo\)"` did not match the same string, despite the general rule that subexpressions and entire BREs match the same strings. To increase consensus, POSIX.1-2024 has allowed an extension on some implementations to treat these two cases in the same way by declaring that anchoring *may* occur at the beginning or end of a subexpression. Therefore, portable BREs that require a literal <circumflex> at the beginning or a <dollar-sign> at the end of a subexpression must escape them. Note that a BRE such as `"a\(^bc\)"` will either match `"a^bc"` or nothing on different systems under the rules.

ERE anchoring has been different from BRE anchoring in all historical systems. An unescaped anchor character has never matched its literal counterpart outside a bracket expression. Some implementations treated `"foo$bar"` as a valid expression that never matched anything; others treated it as invalid. POSIX.1-2024 mandates the former, valid unmatched behavior.

Some implementations have extended the BRE syntax to add alternation. For example, the subexpression `"\(foo$\|bar\)"` would match either `"foo"` at the end of the string or `"bar"` anywhere. The extension is triggered by the use of the undefined `"\|"` sequence. Because the BRE is undefined for portable scripts, the extending system is free to make other assumptions, such that the `'$'` represents the end-of-line anchor in the middle of a subexpression. If it were not for the extension, the `'$'` would match a literal <dollar-sign> under the rules.

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0049 [595] is applied.

Austin Group Defect 1546 is applied, adding optional support for `"\?"`, `"\+"`, and `"\|"`.

Austin Group Defect 1579 is applied, eliminating an inconsistency between the list items relating to <circumflex> and <dollar-sign>.

A.9.4 Extended Regular Expressions

As with BREs, the standard developers decided to make the interpretation of escaped ordinary characters undefined.

The <right-parenthesis> is not listed as an ERE special character because it is only special in the context of a preceding <left-parenthesis>. If found without a preceding <left-parenthesis>, the <right-parenthesis> has no special meaning.

The interval expression, `"{m,n}"`, has been added to EREs. Historically, the interval expression has only been supported in some ERE implementations. The standard developers estimated that the addition of interval expressions to EREs would not decrease consensus and would also make BREs more of a subset of EREs than in many historical implementations.

It was suggested that, in addition to interval expressions, back-references (`'\n'`) should also be added to EREs. This was rejected by the standard developers as likely to decrease consensus.

In historical implementations, multiple duplication symbols are usually interpreted from left to right and treated as additive. As an example, `"a+*b"` matches zero or more instances of `'a'` followed by a `'b'`. In POSIX.1-2024, multiple duplication symbols are undefined; that is, they cannot be relied upon for conforming applications. One reason for this is to provide some scope for future enhancements.

The precedence of operations differs between EREs and those in *lex*; in *lex*, for historical reasons, interval expressions have a lower precedence than concatenation.

Austin Group Defect 1139 is applied, making minor editorial changes to several subsections of this section and changing them to require that, when not inside a bracket expression, `"\]"` matches `']'` and `"\}"` matches `'}'`.

A.9.4.1 EREs Matching a Single Character or Collating Element

There is no additional rationale provided for this section.

A.9.4.2 ERE Ordinary Characters

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0050 [554] is applied.

A.9.4.3 ERE Special Characters

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0050 [554] is applied.

A.9.4.4 Periods in EREs

There is no additional rationale provided for this section.

A.9.4.5 ERE Bracket Expression

There is no additional rationale provided for this section.

127617 *A.9.4.6 EREs Matching Multiple Characters*

127618 Austin Group Defects 793 and 1329 are applied, adding the repetition modifier `'?'` and the
127619 REG_MINIMAL flag.

127620 *A.9.4.7 ERE Alternation*

127621 There is no additional rationale provided for this section.

127622 *A.9.4.8 ERE Precedence*

127623 There is no additional rationale provided for this section.

127624 *A.9.4.9 ERE Expression Anchoring*

127625 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0051 [595] is applied.

127626 ### A.9.5 Regular Expression Grammar

127627 The grammars are intended to represent the range of acceptable syntaxes available to
127628 conforming applications. There are instances in the text where undefined constructs are
127629 described; as explained previously, these allow implementation extensions. There is no intended
127630 requirement that an implementation extension must somehow fit into the grammars shown
127631 here.

127632 The BRE grammar does not permit L_ANCHOR or R_ANCHOR inside `"\("` and `"\)"` (which
127633 implies that `'^'` and `'$'` are ordinary characters). This reflects the semantic limits on the
127634 application, as noted in XBD Section 9.3.8 (on page 186). Implementations are permitted to
127635 extend the language to interpret `'^'` and `'$'` as anchors in these locations, and as such,
127636 conforming applications cannot use unescaped `'^'` and `'$'` in positions inside `"\("` and `"\)"`
127637 that might be interpreted as anchors.

127638 The ERE grammar does not permit several constructs that XBD Section 9.4.2 (on page 187) and
127639 Section 9.4.3 (on page 188) specify as having undefined results:

127640 - ORD_CHAR preceded by <backslash>
127641 - *ERE_dupl_symbol*(s) appearing first in an ERE, or immediately following `'|'`, `'^'`, or `'('`
127642 - `'{'` not part of a valid *ERE_dupl_symbol*
127643 - `'|'` appearing first or last in an ERE, or immediately following `'|'` or `'('`, or
127644 immediately preceding `')'`

127645 Implementations are permitted to extend the language to allow these. Conforming applications
127646 cannot use such constructs.

127647 *A.9.5.1 BRE/ERE Grammar Lexical Conventions*

127648 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0052 [554] is applied.

127649 Austin Group Defect 1139 is applied, updating **QUOTED_CHAR** to add `\]` to the BRE list and
127650 add `\]` and `\}` to the ERE list, and changing ``outside bracket expressions'' to ``except inside
127651 bracket expressions''.

127652 Austin Group Defect 1546 is applied, adding optional support for `\?`, `\+`, and `\|` in BREs.

127653 *A.9.5.2 RE and Bracket Expression Grammar*

127654 The removal of the *Back_open_paren Back_close_paren* option from the *nondupl_RE* specification is
127655 the result of PASC Interpretation 1003.2-92 #43 submitted for the ISO POSIX-2: 1993 standard.
127656 Although the grammar required support for null subexpressions, this section does not describe
127657 the meaning of, and historical practice did not support, this construct.

127658 Austin Group Defect 1546 is applied, adding optional support for `\?`, `\+`, and `\|` in BREs.

127659 *A.9.5.3 ERE Grammar*

127660 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0052 [554] and XBD/TC2-2008/0053
127661 [916] are applied.

127662 A.10 Directory Structure and Devices

127663 A.10.1 Directory Structure and Files

127664 A description of the historical **/usr/tmp** was omitted, removing any concept of differences in
127665 emphasis between the **/** and **/usr** directories. The descriptions of **/bin**, **/usr/bin**, **/lib**, and **/usr/lib**
127666 were omitted because they are not useful for applications. In an early draft, a distinction was
127667 made between system and application directory usage, but this was not found to be useful.

127668 The directories **/** and **/dev** are included because the notion of a hierarchical directory structure is
127669 key to other information presented elsewhere in POSIX.1-2024. In early drafts, it was argued that
127670 special devices and temporary files could conceivably be handled without a directory structure
127671 on some implementations. For example, the system could treat the characters **"/tmp"** as a
127672 special token that would store files using some non-POSIX file system structure. This notion was
127673 rejected by the standard developers, who required that all the files in this section be
127674 implemented via POSIX file systems.

127675 The **/tmp** directory is retained in POSIX.1-2024 to accommodate historical applications that
127676 assume its availability. Implementations are encouraged to provide suitable directory names in
127677 the environment variable *TMPDIR* and applications are encouraged to use the contents of
127678 *TMPDIR* for creating temporary files.

127679 The standard files **/dev/null** and **/dev/tty** are required to be both readable and writable to allow
127680 applications to have the intended historical access to these files.

127681 The standard file **/dev/console** has been added for alignment with the Single UNIX
127682 Specification.

127683 A.10.2 Output Devices and Terminal Types

127684 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/17 is applied, making it clear that the
127685 requirements for documenting terminal support are in the system documentation.

127686 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0054 [967] is applied.

A.11 General Terminal Interface

If the implementation does not support this interface on any device types, it should behave as if it were being used on a device that is not a terminal device (in most cases *errno* will be set to [ENOTTY] on return from functions defined by this interface). This is based on the fact that many applications are written to run both interactively and in some non-interactive mode, and they adapt themselves at runtime. Requiring that they all be modified to test an environment variable to determine whether they should try to adapt is unnecessary. On a system that provides no general terminal interface, providing all the entry points as stubs that return [ENOTTY] (or an equivalent, as appropriate) has the same effect and requires no changes to the application.

Although the needs of both interface implementors and application developers were addressed throughout POSIX.1-2024, this section pays more attention to the needs of the latter. This is because, while many aspects of the programming interface can be hidden from the user by the application developer, the terminal interface is usually a large part of the user interface. Although to some extent the application developer can build missing features or work around inappropriate ones, the difficulties of doing that are greater in the terminal interface than elsewhere. For example, efficiency prohibits the average program from interpreting every character passing through it in order to simulate character erase, line kill, and so on. These functions should usually be done by the operating system, possibly at the interrupt level.

The *tc**( ) functions were introduced as a way of avoiding the problems inherent in the traditional *ioctl*( ) function and in variants of it that were proposed. For example, *tcsetattr*( ) is specified in place of the use of the TCSETA *ioctl*( ) command function. This allows specification of all the arguments in a manner consistent with the ISO C standard unlike the varying third argument of *ioctl*( ), which is sometimes a pointer (to any of many different types) and sometimes an **int**.

The advantages of this new method include:

- It allows strict type checking.
- The direction of transfer of control data is explicit.
- Portable capabilities are clearly identified.
- The need for a general interface routine is avoided.
- Size of the argument is well-defined (there is only one type).

The disadvantages include:

- No historical implementation used the new method.
- There are many small routines instead of one general-purpose one.
- The historical parallel with *fcntl*( ) is broken.

The issue of modem control was excluded from POSIX.1-2024 on the grounds that:

- It was concerned with setting and control of hardware timers.
- The appropriate timers and settings vary widely internationally.
- Feedback from European computer manufacturers indicated that this facility was not consistent with European needs and that specification of such a facility was not a requirement for portability.

A.11.1 Interface Characteristics

A.11.1.1 Opening a Terminal Device File

The O_TTY_INIT flag for *open*() has been added to POSIX.1-2024 to solve a problem encountered by applications written for earlier versions of this standard which need to open a modem or similar device and initialize all of the parameter settings. Using the *tcgetattr*()-modify-*tcsetattr*() method mandated by the standard could result in non-conforming behavior if the device had previously been used with non-conforming parameter settings, on implementations which do not reset the parameter settings in between the last close of the device by one application and the first open by another application. To avoid this problem, some application developers were resorting to using *memset*() to zero the **termios** structure before setting all of the standard parameters, but this risks non-conforming behavior on systems where some non-standard parameter needs a non-zero value in order for the terminal to behave in a conforming manner.

On systems which do reset the parameter settings to defaults between uses of a terminal device, it is expected that either O_TTY_INIT will have the value zero or *open*(*ttypath*, O_RDWR|O_TTY_INIT) will do nothing additional.

The standard developers considered an alternative solution of a special *fildes* argument for the *tcgetattr*() call to obtain default parameters. However, this would not be adequate if a system supports several different types of terminal device and the default settings need to differ between the different types. With the O_TTY_INIT open flag, the implementor can determine which device type is being opened.

The standard developers also considered a special POSIX_TTY_INIT value for the **termios** structure used in *tcsetattr*(), which would reset the values if used immediately after an *open*() call. However, it was felt that this would lead to confusion amongst application developers who wanted to reset the parameters at other points, and implementations might diverge.

Austin Group Defect 1466 is applied, changing the terminology used for pseudo-terminal devices.

A.11.1.2 Process Groups

There is a potential race when the members of the foreground process group on a terminal leave that process group, either by exit or by changing process groups. After the last process exits the process group, but before the foreground process group ID of the terminal is changed (usually by a job control shell), it would be possible for a new process to be created with its process ID equal to the terminal's foreground process group ID. That process might then become the process group leader and accidentally be placed into the foreground on a terminal that was not necessarily its controlling terminal. As a result of this problem, the controlling terminal is defined to not have a foreground process group during this time.

The cases where a controlling terminal has no foreground process group occur when all processes in the foreground process group either terminate and are waited for or join other process groups via *setpgid*() or *setsid*(). If the process group leader terminates, this is the first case described; if it leaves the process group via *setpgid*(), this is the second case described (a process group leader cannot successfully call *setsid*()). When one of those cases causes a controlling terminal to have no foreground process group, it has two visible effects on applications. The first is the value returned by *tcgetpgrp*(). The second (which occurs only in the case where the process group leader terminates) is the sending of signals in response to special

input characters. The intent of POSIX.1-2024 is that no process group be wrongly identified as the foreground process group by *tcgetpgrp*() or unintentionally receive signals because of placement into the foreground.

In 4.3 BSD, the old process group ID continues to be used to identify the foreground process group and is returned by the function equivalent to *tcgetpgrp*(). In that implementation it is possible for a newly created process to be assigned the same value as a process ID and then form a new process group with the same value as a process group ID. The result is that the new process group would receive signals from this terminal for no apparent reason, and POSIX.1-2024 precludes this by forbidding a process group from entering the foreground in this way. It would be more direct to place part of the requirement made by the last sentence under *fork*(), but there is no convenient way for that section to refer to the value that *tcgetpgrp*() returns, since in this case there is no process group and thus no process group ID.

One possibility for a conforming implementation is to behave similarly to 4.3 BSD, but to prevent this reuse of the ID, probably in the implementation of *fork*(), as long as it is in use by the terminal.

Another possibility is to recognize when the last process stops using the terminal's foreground process group ID, which is when the process group lifetime ends, and to change the terminal's foreground process group ID to a reserved value that is never used as a process ID or process group ID. (See the definition of *process group lifetime* in the definitions section.) The process ID can then be reserved until the terminal has another foreground process group.

The 4.3 BSD implementation permits the leader (and only member) of the foreground process group to leave the process group by calling the equivalent of *setpgid*() and to later return, expecting to return to the foreground. There are no known application needs for this behavior, and POSIX.1-2024 neither requires nor forbids it (except that it is forbidden for session leaders) by leaving it unspecified.

A.11.1.3 The Controlling Terminal

POSIX.1-2024 does not specify a mechanism by which to allocate a controlling terminal. This is normally done by a system utility (such as *getty*) and is considered an administrative feature outside the scope of POSIX.1-2024.

Historical implementations allocate controlling terminals on certain *open*() calls. Since *open*() is part of POSIX.1, its behavior had to be dealt with. The traditional behavior is not required because it is not very straightforward or flexible for either implementations or applications. However, because of its prevalence, it was not practical to disallow this behavior either. Thus, a mechanism was standardized to ensure portable, predictable behavior in *open*().

Some historical implementations deallocate a controlling terminal on the last system-wide close. This behavior in neither required nor prohibited. Even on implementations that do provide this behavior, applications generally cannot depend on it due to its system-wide nature.

A.11.1.4 Terminal Access Control

The access controls described in this section apply only to a process that is accessing its controlling terminal. A process accessing a terminal that is not its controlling terminal is effectively treated the same as a member of the foreground process group. While this may seem unintuitive, note that these controls are for the purpose of job control, not security, and job control relates only to the controlling terminal of a process. Normal file access permissions handle security.

If the process calling *read*() or *write*() is in a background process group that is orphaned, it is not

desirable to stop the process group, as it is no longer under the control of a job control shell that could put it into the foreground again. Accordingly, calls to *read*( ) or *write*( ) functions by such processes receive an immediate error return. This is different from 4.2 BSD, which kills orphaned processes that receive terminal stop signals.

The foreground/background/orphaned process group check performed by the terminal driver must be repeatedly performed until the calling process moves into the foreground or until the process group of the calling process becomes orphaned. That is, when the terminal driver determines that the calling process is in the background and should receive a job control signal, it sends the appropriate signal (SIGTTIN or SIGTTOU) to every process in the process group of the calling process and then it allows the calling process to immediately receive the signal. The latter is typically performed by blocking the process so that the signal is immediately noticed. Note, however, that after the process finishes receiving the signal and control is returned to the driver, the terminal driver must re-execute the foreground/background/orphaned process group check. The process may still be in the background, either because it was continued in the background by a job control shell, or because it caught the signal and did nothing.

The terminal driver repeatedly performs the foreground/background/orphaned process group checks whenever a process is about to access the terminal. In the case of *write*( ) or the control *tc**( ) functions, the check is performed at the entry of the function. In the case of *read*( ), the check is performed not only at the entry of the function, but also after blocking the process to wait for input characters (if necessary). That is, once the driver has determined that the process calling the *read*( ) function is in the foreground, it attempts to retrieve characters from the input queue. If the queue is empty, it blocks the process waiting for characters. When characters are available and control is returned to the driver, the terminal driver must return to the repeated foreground/background/orphaned process group check again. The process may have moved from the foreground to the background while it was blocked waiting for input characters.

Austin Group Defect 1151 is applied, adding *tcsetwinsize*( ).

A.11.1.5 Input Processing and Reading Data

There is no additional rationale provided for this section.

A.11.1.6 Canonical Mode Input Processing

The term ``character'' is intended here. ERASE should erase the last character, not the last byte. In the case of multi-byte characters, these two may be different.

4.3 BSD has a WERASE character that erases the last ``word'' typed (but not any preceding <blank> or <tab> characters). A word is defined as a sequence of non-<blank> characters, with <tab> characters counted as <blank> characters. Like ERASE, WERASE does not erase beyond the beginning of the line. This WERASE feature has not been specified in POSIX.1 because it is difficult to define in the international environment. It is only useful for languages where words are delimited by <blank> characters. In some ideographic languages, such as Japanese and Chinese, words are not delimited at all. The WERASE character should presumably go back to the beginning of a sentence in those cases; practically, this means it would not be used much for those languages.

It should be noted that there is a possible inherent deadlock if the application and implementation conflict on the value of {MAX_CANON}. With ICANON set (if IXOFF is enabled) and more than {MAX_CANON} characters transmitted without a <linefeed>, transmission will be stopped, the <linefeed> (or <carriage-return> when ICRLF is set) will never arrive, and the *read*( ) will never be satisfied.

An application should not set IXOFF if it is using canonical mode unless it knows that (even in the face of a transmission error) the conditions described previously cannot be met or unless it is prepared to deal with the possible deadlock in some other way, such as timeouts.

It should also be noted that this can be made to happen in non-canonical mode if the trigger value for sending IXOFF is less than VMIN and VTIME is zero.

A.11.1.7 Non-Canonical Mode Input Processing

Some points to note about MIN and TIME:

1. The interactions of MIN and TIME are not symmetric. For example, when MIN>0 and TIME=0, TIME has no effect. However, in the opposite case where MIN=0 and TIME>0, both MIN and TIME play a role in that MIN is satisfied with the receipt of a single character.
2. Also note that in case A (MIN>0, TIME>0), TIME represents an inter-character timer, while in case C (MIN=0, TIME>0), TIME represents a read timer.

These two points highlight the dual purpose of the MIN/TIME feature. Cases A and B, where MIN>0, exist to handle burst-mode activity (for example, file transfer programs) where a program would like to process at least MIN characters at a time. In case A, the inter-character timer is activated by a user as a safety measure; in case B, it is turned off.

Cases C and D exist to handle single-character timed transfers. These cases are readily adaptable to screen-based applications that need to know if a character is present in the input queue before refreshing the screen. In case C, the read is timed; in case D, it is not.

Another important note is that MIN is always just a minimum. It does not denote a record length. That is, if a program does a read of 20 bytes, MIN is 10, and 25 characters are present, 20 characters are returned to the user. In the special case of MIN=0, this still applies: if more than one character is available, they all will be returned immediately.

A.11.1.8 Writing Data and Output Processing

There is no additional rationale provided for this section.

A.11.1.9 Special Characters

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0055 [745] is applied.

A.11.1.10 Modem Disconnect

There is no additional rationale provided for this section.

A.11.1.11 Closing a Terminal Device File

POSIX.1-2024 does not specify that a *close*() on a terminal device file include the equivalent of a call to *tcflow*(*fd*,TCOON).

An implementation that discards output at the time *close*() is called after reporting the return value to the *write*() call that data was written does not conform with POSIX.1-2024. An application has functions such as *tcdrain*(), *tcflush*(), and *tcflow*() available to obtain the detailed behavior it requires with respect to flushing of output.

At the time of the last close on a terminal device, an application relinquishes any ability to exert

127900 flow control via *tcflow*( ).

127901 A.11.2 Parameters that Can be Set

127902 *A.11.2.1 The termios Structure*

127903 This structure is part of an interface that, in general, retains the historic grouping of flags.
127904 Although a more optimal structure for implementations may be possible, the degree of change
127905 to applications would be significantly larger.

127906 *A.11.2.2 Input Modes*

127907 Some historical implementations treated a long break as multiple events, as many as one per
127908 character time. The wording in POSIX.1 explicitly prohibits this.

127909 Although the ISTRIP flag is normally superfluous with today's terminal hardware and software,
127910 it is historically supported. Therefore, applications may be using ISTRIP, and there is no
127911 technical problem with supporting this flag. Also, applications may wish to receive only 7-bit
127912 input bytes and may not be connected directly to the hardware terminal device (for example,
127913 when a connection traverses a network).

127914 Also, there is no requirement in general that the terminal device ensures that high-order bits
127915 beyond the specified character size are cleared. ISTRIP provides this function for 7-bit
127916 characters, which are common.

127917 In dealing with multi-byte characters, the consequences of a parity error in such a character, or
127918 in an escape sequence affecting the current character set, are beyond the scope of POSIX.1 and
127919 are best dealt with by the application processing the multi-byte characters.

127920 *A.11.2.3 Output Modes*

127921 POSIX.1 does not describe post-processing of output to a terminal or detailed control of that
127922 from a conforming application. (That is, translation of <newline> to <carriage-return> followed
127923 by <linefeed> or <tab> processing.) There is nothing that a conforming application should do to
127924 its output for a terminal because that would require knowledge of the operation of the terminal.
127925 It is the responsibility of the operating system to provide post-processing appropriate to the
127926 output device, whether it is a terminal or some other type of device.

127927 Extensions to POSIX.1 to control the type of post-processing already exist and are expected to
127928 continue into the future. The control of these features is primarily to adjust the interface between
127929 the system and the terminal device so the output appears on the display correctly. This should
127930 be set up before use by any application.

127931 In general, both the input and output modes should not be set absolutely, but rather modified
127932 from the inherited state.

127933 *A.11.2.4 Control Modes*

127934 This section could be misread that the symbol ``CSIZE'' is a title in the **termios** *c_cflag* field.
127935 Although it does serve that function, it is also a required symbol, as a literal reading of POSIX.1
127936 (and the caveats about typography) would indicate.

127937 *A.11.2.5 Local Modes*

127938 Non-canonical mode is provided to allow fast bursts of input to be read efficiently while still
127939 allowing single-character input.

127940 The ECHONL function historically has been in many implementations. Since there seems to be
127941 no technical problem with supporting ECHONL, it is included in POSIX.1 to increase consensus.

127942 The alternate behavior possible when ECHOK or ECHOE are specified with ICANON is
127943 permitted as a compromise depending on what the actual terminal hardware can do. Erasing
127944 characters and lines is preferred, but is not always possible.

127945 *A.11.2.6 Special Control Characters*

127946 Permitting VMIN and VTIME to overlap with VEOF and VEOL was a compromise for historical
127947 implementations. Only when backwards-compatibility of object code is a serious concern to an
127948 implementor should an implementation continue this practice. Correct applications that work
127949 with the overlap (at the source level) should also work if it is not present, but not the reverse.

127950 A.12 Utility Conventions

127951 A.12.1 Utility Argument Syntax

127952 The standard developers considered that recent trends toward diluting the SYNOPSIS sections
127953 of historical reference pages to the equivalent of:

127954 `command [`*`options`*`][`*`operands`*`]`

127955 were a disservice to the reader. Therefore, considerable effort was placed into rigorous
127956 definitions of all the command line arguments and their interrelationships. The relationships
127957 depicted in the synopses are normative parts of POSIX.1-2024; this information is sometimes
127958 repeated in textual form, but that is only for clarity within context.

127959 The use of ``undefined'' for conflicting argument usage and for repeated usage of the same
127960 option is meant to prevent conforming applications from using conflicting arguments or
127961 repeated options unless specifically allowed (as is the case with *ls*, which allows simultaneous,
127962 repeated use of the **–C**, **–l**, and **–1** options). Many historical implementations will tolerate this
127963 usage, choosing either the first or the last applicable argument. This tolerance can continue, but
127964 conforming applications cannot rely upon it. (Other implementations may choose to print usage
127965 messages instead.)

127966 The use of ``undefined'' for conflicting argument usage also allows an implementation to make
127967 reasonable extensions to utilities where the implementor considers mutually-exclusive options
127968 according to POSIX.1-2024 to have a sensible meaning and result.

127969 POSIX.1-2024 does not define the result of a command when an option-argument or operand is
127970 not followed by ellipses and the application specifies more than one of that option-argument or

127971 operand. This allows an implementation to define valid (although non-standard) behavior for
127972 the utility when more than one such option or operand is specified.

127973 The requirements for option-arguments are summarized as follows:

	SYNOPSIS Shows:	
	`-a arg`	`-c[arg]`
Conforming application uses:	`-a arg`	`-carg` or `-c`
System supports:	`-a arg` and `-aarg`	`-carg` and `-c`
Non-conforming applications may use:	`-aarg`	N/A

127979 Earlier versions of this standard included obsolescent syntax which showed some options with
127980 (mandatory) adjacent option-arguments in the SYNOPSIS for some utilities. These have since
127981 been removed. For all options with mandatory option-arguments, the SYNOPSIS now shows
127982 <blank> characters between the option and the option-argument; however, historical usage has
127983 not been consistent in this area; therefore, <blank> characters are required to be used by
127984 conforming applications and to be handled by all implementations, but implementations are
127985 also required to handle an adjacent option-argument in order to preserve backwards-
127986 compatibility for old scripts. One of the justifications for selecting the multiple-argument
127987 method was that the single-argument case is inherently ambiguous when the option-argument
127988 can legitimately be a null string.

127989 POSIX.1-2024 explicitly states that digits are permitted as operands and option-arguments. The
127990 lower and upper bounds for the values of the numbers used for operands and option-arguments
127991 were derived from the ISO C standard values for {LONG_MIN} and {LONG_MAX}. The
127992 requirement on the standard utilities is that numbers in the specified range do not cause a
127993 syntax error, although the specification of a number need not be semantically correct for a
127994 particular operand or option-argument of a utility. For example, the specification of:

127995 `dd obs=3000000000`

127996 would yield undefined behavior for the application and could be a syntax error because the
127997 number 3 000 000 000 is outside of the range –2 147 483 647 to +2 147 483 647. On the other hand:

127998 `dd obs=2000000000`

127999 may cause some error, such as ``blocksize too large'', rather than a syntax error.

128000 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0056 [584] and XBD/TC2-2008/0057
128001 [813] are applied.

128002 Austin Group Defect 1062 is applied, correcting the spacing in some example SYNOPSIS lines.

128003 A.12.2 Utility Syntax Guidelines

128004 This section is based on the rules listed in the SVID. It was included for two reasons:

128005 1. The individual utility descriptions in XCU Chapter 3 (on page 2573) needed a set of
128006 common (although not universal) actions on which they could anchor their descriptions
128007 of option and operand syntax. Most of the standard utilities actually do use these
128008 guidelines, and many of their historical implementations use the *getopt*( ) function for
128009 their parsing. Therefore, it was simpler to cite the rules and merely identify exceptions.

128010 2. Developers of conforming applications need suggested guidelines if the POSIX
128011 community is to avoid the chaos of historical UNIX system command syntax.

128012 It is recommended that all *future* utilities and applications use these guidelines to enhance ``user

128013 portability''. The fact that some historical utilities could not be changed (to avoid breaking
128014 historical applications) should not deter this future goal.

128015 The voluntary nature of the guidelines is highlighted by repeated uses of the word *should*
128016 throughout. This usage should not be misinterpreted to imply that utilities that claim
128017 conformance in their OPTIONS sections do not always conform.

128018 Guidelines 1 and 2 encourage utility writers to use only characters from the portable character
128019 set because use of locale-specific characters may make the utility inaccessible from other locales.
128020 Use of uppercase letters is discouraged due to problems associated with porting utilities to
128021 systems that do not distinguish between uppercase and lowercase characters in filenames. Use
128022 of non-alphanumeric characters is discouraged due to the number of utilities that treat non-
128023 alphanumeric characters in ``special'' ways depending on context (such as the shell using white-
128024 space characters to delimit arguments, various quote characters for quoting, the <dollar-sign> to
128025 introduce variable expansion, etc.).

128026 In XCU Section 2.9.1 (on page 2500), it is further stated that a command used in the Shell
128027 Command Language cannot be named with a trailing <colon>.

128028 Guideline 3 was changed to allow alphanumeric characters (letters and digits) from the
128029 character set to allow compatibility with historical usage. Historical practice allows the use of
128030 digits wherever practical, and there are no portability issues that would prohibit the use of
128031 digits. In fact, from an internationalization viewpoint, digits (being non-language-dependent)
128032 are preferable over letters (a **–2** is intuitively self-explanatory to any user, while in the **–f** *filename*
128033 the letter `'f'` is a mnemonic aid only to speakers of Latin-based languages where ``filename''
128034 happens to translate to a word that begins with `'f'`. Since Guideline 3 still retains the word
128035 ``single'', multi-digit options are not allowed. Instances of historical utilities that used them have
128036 been marked obsolescent, with the numbers being changed from option names to option-
128037 arguments.

128038 It was difficult to achieve a satisfactory solution to the problem of name space in option
128039 characters. When the standard developers desired to extend the historical *cc* utility to accept
128040 ISO C standard programs, they found that all of the portable alphabet was already in use by
128041 various vendors. Thus, they had to devise a new name, *c89* (subsequently superseded by *c99*
128042 and now by *c17*), rather than something like *cc* **–X**. There were suggestions that implementors
128043 be restricted to providing extensions through various means (such as using a <plus-sign> as the
128044 option delimiter or using option characters outside the alphanumeric set) that would reserve all
128045 of the remaining alphanumeric characters for future POSIX standards. These approaches were
128046 resisted because they lacked the historical style of UNIX systems. Furthermore, if a vendor-
128047 provided option should become commonly used in the industry, it would be a candidate for
128048 standardization. It would be desirable to standardize such a feature using historical practice for
128049 the syntax (the semantics can be standardized with any syntax). This would not be possible if
128050 the syntax was one reserved for the vendor. However, since the standardization process may
128051 lead to minor changes in the semantics, it may prove to be better for a vendor to use a syntax
128052 that will not be affected by standardization.

128053 Guideline 8 includes the concept of <comma>-separated lists in a single argument. It is up to the
128054 utility to parse such a list itself because *getopt*( ) just returns the single string. This situation was
128055 retained so that certain historical utilities would not violate the guidelines. Applications
128056 preparing for international use should be aware of an occasional problem with
128057 <comma>-separated lists: in some locales, the <comma> is used as the radix character. Thus, if
128058 an application is preparing operands for a utility that expects a <comma>-separated list, it
128059 should avoid generating non-integer values through one of the means that is influenced by
128060 setting the *LC_NUMERIC* variable (such as *awk*, *bc*, *printf*, or *printf*( )).

128061 Unless explicitly stated otherwise in the utility description, Guideline 9 requires applications to

put options before operands, and requires utilities to accept any such usage without misinterpreting operands as options. For example, if an implementation of the *printf* utility supports a **–e** option as an extension, the command:

```
printf %s -e
```

must output the string `"-e"` without interpreting the **–e** as an option. Similarly, the command:

```
ls myfile -l
```

must interpret the **–l** argument as a second file operand, not as a **–l** option.

Applications calling any utility with a first operand starting with `'-'` should usually specify **––**, as indicated by Guideline 10, to mark the end of the options. This is true even if the SYNOPSIS in the Shell and Utilities volume of POSIX.1-2024 does not specify any options; implementations may provide options as extensions to the Shell and Utilities volume of POSIX.1-2024. The standard utilities that do not support Guideline 10 indicate that fact in the OPTIONS section of the utility description.

Guideline 7 allows any string to be an option-argument; an option-argument can begin with any character, can be – or **––**, and can be an empty string. For example, the commands *pr* **–h** **–**, *pr* **–h** **––**, *pr* **–h** **–d**, *pr* **–h** **+2**, and *pr* **–h** **''** contain the option-arguments **–**, **––**, **–d**, **+2**, and an empty string, respectively. Conversely, the command *pr* **–h** **–– –d** treats **–d** as an option, not as an argument, because the **––** is an option-argument here, not a delimiter.

Guideline 11 was modified to clarify that the order of different options should not matter relative to one another. However, the order of repeated options that also have option-arguments may be significant; therefore, such options are required to be interpreted in the order that they are specified. The *make* utility is an instance of a historical utility that uses repeated options in which the order is significant. Multiple files are specified by giving multiple instances of the **–f** option; for example:

```
make -f common_header -f specific_rules target
```

Guideline 13 does not imply that all of the standard utilities automatically accept the operand `'-'` to mean standard input or output, nor does it specify the actions of the utility upon encountering multiple `'-'` operands. It simply says that, by default, `'-'` operands are not used for other purposes in the file reading or writing (but not when using *stat*( ), *unlink*( ), *touch*, and so on) utilities. In earlier versions of this standard, all information concerning actual treatment of the `'-'` operand is found in the individual utility sections. Many implementations, however, treated `'-'` as standard input or output and many applications depended on this behavior even though it was not standard. This behavior is now implementation-defined. Portable applications should not use `'-'` to mean standard input or output unless it is explicitly stated to do so in the utility description and they should always use `'./-'` if they intend to refer to a file named – in the current working directory.

Guideline 14 is intended to prohibit implementations that would treat the command *ls* **–l –d** as if it were *ls* **–– –l –d** or *ls* **–l –– –d**.

The standard permits implementations to have extensions that violate the Utility Syntax Guidelines so long as when the utility is used in line with the forms defined by the standard it follows the Utility Syntax Guidelines. Thus, head*–42*file and *ls*––help are permitted extensions. The intent is to allow extensions so long as the standard form is accepted and follows the guidelines.

An area of concern was that as implementations mature, implementation-defined utilities and implementation-defined utility options will result. The idea was expressed that there needed to be a standard way, say an environment variable or some such mechanism, to identify implementation-defined utilities separately from standard utilities that may have the same

name. It was decided that there already exist several ways of dealing with this situation and that it is outside of the scope to attempt to standardize in the area of non-standard items. A method that exists on some historical implementations is the use of the so-called **/local/bin** or **/usr/local/bin** directory to separate local or additional copies or versions of utilities. Another method that is also used is to isolate utilities into completely separate domains. Still another method to ensure that the desired utility is being used is to request the utility by its full pathname. There are many approaches to this situation; the examples given above serve to illustrate that there is more than one.

A.13 Namespace and Future Directions

Austin Group Defect 1071 is applied, adding this chapter.

A.14 Headers

A.14.1 Format of Entries

Each header reference page has a common layout of sections describing the interface. This layout is similar to the manual page or ``man'' page format shipped with most UNIX systems, and each header has sections describing the SYNOPSIS and DESCRIPTION. These are the two sections that relate to conformance.

Additional sections are informative, and add considerable information for the application developer. APPLICATION USAGE sections provide additional caveats, issues, and recommendations to the developer. RATIONALE sections give additional information on the decisions made in defining the interface.

FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in the future, and often cautions the developer to architect the code to account for a change in this area. Note that a future directions statement should not be taken as a commitment to adopt a feature or interface in the future.

The CHANGE HISTORY section describes when the interface was introduced, and how it has changed.

Option labels and margin markings in the page can be useful in guiding the application developer.

A.14.2 Removed Headers in Issue 8

The headers removed in Issue 8 (from the Issue 7 base document) are as follows:

Removed Headers in Issue 8	
<stropts.h>	**<trace.h>**
<ulimit.h>	**<utime.h>**

Rationale (Informative)

Part B:

System Interfaces

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

128146 *Appendix B*

128147 # *Rationale for System Interfaces*

128148 ## B.1 Introduction

128149 ### B.1.1 Change History

128150 The change history is provided as an informative section, to track changes from earlier versions
128151 of this standard.

128152 The following sections describe changes made to the System Interfaces volume of POSIX.1-2024
128153 since Issue 7 of the base document. The CHANGE HISTORY section for each entry details the
128154 technical changes that have been made in Issue 5 and later. Changes made before Issue 5 are not
128155 included.

128156 **Changes from Issue 7 to Issue 8 (POSIX.1-2024)**

128157 The following list summarizes the major changes that were made in the System Interfaces
128158 volume of POSIX.1-2024 from Issue 7 to Issue 8:

128159 • The Open Group Standard, 2021, Additional APIs for the Base Specifications Issue 8, Part 1
128160 is incorporated.

128161 • The Open Group Standard, 2022, Additional APIs for the Base Specifications Issue 8, Part 2
128162 is incorporated.

128163 • IEEE Std 1003.26-2003 is incorporated.

128164 • Existing functionality is aligned with the ISO/IEC 9899: 2018 standard.

128165 • New functionality from the ISO/IEC 9899: 2018 standard is incorporated.

128166 • Austin Group defect reports and IEEE Interpretations against IEEE Std 1003.1 are applied.

128167 • The Open Group corrigenda and resolutions are applied.

128168 • Features, marked obsolescent in the base document, have been considered for removal in
128169 this version.

128170 • The Device Control option is added.

128171 • The IEC 60559 Complex Floating-Point option is added.

New Features in Issue 8

The functions first introduced in Issue 8 (over the Issue 7 base document) are as follows:

New Functions in Issue 8		
_Fork()	*c32rtomb*()	*posix_close*()
aligned_alloc()	*call_once*()	*posix_devctl*()
at_quick_exit()	*cnd_broadcast*()	*posix_getdents*()
atomic_compare_exchange_strong()	*cnd_destroy*()	*ppoll*()
atomic_compare_exchange_strong_explicit()	*cnd_init*()	*pthread_cond_clockwait*()
atomic_compare_exchange_weak()	*cnd_signal*()	*pthread_mutex_clocklock*()
atomic_compare_exchange_weak_explicit()	*cnd_timedwait*()	*pthread_rwlock_clockrdlock*()
atomic_exchange()	*cnd_wait*()	*pthread_rwlock_clockwrlock*()
atomic_exchange_explicit()	*dcgettext*()	*qsort_r*()
atomic_fetch_add()	*dcgettext_l*()	*quick_exit*()
atomic_fetch_add_explicit()	*dcngettext*()	*reallocarray*()
atomic_fetch_and()	*dcngettext_l*()	*sem_clockwait*()
atomic_fetch_and_explicit()	*dgettext*()	*setresgid*()
atomic_fetch_or()	*dgettext_l*()	*setresuid*()
atomic_fetch_or_explicit()	*dladdr*()	*sig2str*()
atomic_fetch_sub()	*dngettext*()	*str2sig*()
atomic_fetch_sub_explicit()	*dngettext_l*()	*strlcat*()
atomic_fetch_xor()	*getentropy*()	*strlcpy*()
atomic_fetch_xor_explicit()	*getlocalename_l*()	*textdomain*()
atomic_flag_clear()	*getresgid*()	*thrd_create*()
atomic_flag_clear_explicit()	*getresuid*()	*thrd_current*()
atomic_flag_test_and_set()	*gettext*()	*thrd_detach*()
atomic_flag_test_and_set_explicit()	*gettext_l*()	*thrd_equal*()
atomic_init()	*mbrtoc16*()	*thrd_exit*()
atomic_is_lock_free()	*mbrtoc32*()	*thrd_join*()
atomic_load()	*memmem*()	*thrd_sleep*()
atomic_load_explicit()	*mtx_destroy*()	*thrd_yield*()
atomic_signal_fence()	*mtx_init*()	*timespec_get*()
atomic_store()	*mtx_lock*()	*tss_create*()
atomic_store_explicit()	*mtx_timedlock*()	*tss_delete*()
atomic_thread_fence()	*mtx_trylock*()	*tss_get*()
bind_textdomain_codeset()	*mtx_unlock*()	*tss_set*()
bindtextdomain()	*ngettext*()	*wcslcat*()
c16rtomb()	*ngettext_l*()	*wcslcpy*()

The following new headers are introduced in Issue 8:

New Headers in Issue 8		
<devctl.h>	**<stdatomic.h>**	**<uchar.h>**
<libintl.h>	**<stdnoreturn.h>**	
<stdalign.h>	**<threads.h>**	

128214 **Obsolescent Functions in Issue 8**

128215 The base functions moved to obsolescent status in Issue 8 (from the Issue 7 base document) are
128216 as follows:

128217

Obsolescent Base Functions in Issue 8	
128218 *inet_addr*( )	*inet_ntoa*( )

128219 The XSI functions moved to obsolescent status in Issue 8 (from the Issue 7 base document) are as
128220 follows:

128221

Obsolescent XSI Functions in Issue 8	
128222 *encrypt*( )	*setkey*( )

128223 **Removed Functions in Issue 8**

128224 The functions removed in Issue 8 (from the Issue 7 base document) are as follows:

Removed Functions in Issue 8	
_longjmp( )	*posix_trace_eventid_equal*( )
_setjmp( )	*posix_trace_eventid_get_name*( )
_tolower( )	*posix_trace_eventid_open*( )
_toupper( )	*posix_trace_eventset_add*( )
fattach( )	*posix_trace_eventset_del*( )
fdetach( )	*posix_trace_eventset_empty*( )
ftw( )	*posix_trace_eventset_fill*( )
getitimer( )	*posix_trace_eventset_ismember*( )
getmsg( )	*posix_trace_eventtypelist_getnext_id*( )
getpmsg( )	*posix_trace_eventtypelist_rewind*( )
gets( )	*posix_trace_flush*( )
gettimeofday( )	*posix_trace_get_attr*( )
ioctl( )	*posix_trace_get_filter*( )
isascii( )	*posix_trace_get_status*( )
isastream( )	*posix_trace_getnext_event*( )
posix_trace_attr_destroy( )	*posix_trace_open*( )
posix_trace_attr_getclockres( )	*posix_trace_rewind*( )
posix_trace_attr_getcreatetime( )	*posix_trace_set_filter*( )
posix_trace_attr_getgenversion( )	*posix_trace_shutdown*( )
posix_trace_attr_getinherited( )	*posix_trace_start*( )
posix_trace_attr_getlogfullpolicy( )	*posix_trace_stop*( )
posix_trace_attr_getlogsize( )	*posix_trace_timedgetnext_event*( )
posix_trace_attr_getmaxdatasize( )	*posix_trace_trid_eventid_open*( )
posix_trace_attr_getmaxsystemeventsize( )	*posix_trace_trygetnext_event*( )
posix_trace_attr_getmaxusereventsize( )	*pthread_getconcurrency*( )
posix_trace_attr_getname( )	*pthread_setconcurrency*( )
posix_trace_attr_getstreamfullpolicy( )	*putmsg*( )
posix_trace_attr_getstreamsize( )	*putpmsg*( )
posix_trace_attr_init( )	*rand_r*( )
posix_trace_attr_setinherited( )	*setitimer*( )
posix_trace_attr_setlogfullpolicy( )	*setpgrp*( )
posix_trace_attr_setlogsize( )	*sighold*( )
posix_trace_attr_setmaxdatasize( )	*sigignore*( )
posix_trace_attr_setname( )	*siginterrupt*( )
posix_trace_attr_setstreamfullpolicy( )	*sigpause*( )
posix_trace_attr_setstreamsize( )	*sigrelse*( )
posix_trace_clear( )	*sigset*( )
posix_trace_close( )	*tempnam*( )
posix_trace_create( )	*toascii*( )
posix_trace_create_withlog( )	*ulimit*( )
posix_trace_event( )	*utime*( )

B.1.2 Relationship to Other Formal Standards

There is no additional rationale provided for this section.

B.1.3 Format of Entries

Each system interface reference page has a common layout of sections describing the interface. This layout is similar to the manual page or ``man'' page format shipped with most UNIX systems, and each header has sections describing the SYNOPSIS, DESCRIPTION, RETURN VALUE, and ERRORS. These are the four sections that relate to conformance.

Additional sections are informative, and add considerable information for the application developer. EXAMPLES sections provide example usage. APPLICATION USAGE sections provide additional caveats, issues, and recommendations to the developer. RATIONALE sections give additional information on the decisions made in defining the interface.

FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in the future, and often cautions the developer to architect the code to account for a change in this area. Note that a future directions statement should not be taken as a commitment to adopt a feature or interface in the future.

The CHANGE HISTORY section describes when the interface was introduced, and how it has changed.

Option labels and margin markings in the page can be useful in guiding the application developer.

B.2 General Information

B.2.1 Use and Implementation of Interfaces

B.2.1.1 Use and Implementation of Functions

The information concerning the use of functions was adapted from a description in the ISO C standard. Here is an example of how an application program can protect itself from functions that may or may not be macros, rather than true functions:

The *atoi*( ) function may be used in any of several ways:

- By use of its associated header (possibly generating a macro expansion):

```
#include <stdlib.h>
/* ... */
i = atoi(str);
```

- By use of its associated header (assuredly generating a true function call):

```
#include <stdlib.h>
#undef atoi
/* ... */
i = atoi(str);
```


or:

```
#include <stdlib.h>
/* ... */
i = (atoi) (str);
```

- By explicit declaration:

```
extern int atoi (const char *);
/* ... */
i = atoi(str);
```

- By implicit declaration:

```
/* ... */
i = atoi(str);
```

(Assuming no function prototype is in scope. This is not allowed by the ISO C standard for functions with variable arguments; furthermore, parameter type conversion ``widening'' is subject to different rules in this case.)

Note that the ISO C standard reserves names starting with `'_'` for the compiler. Therefore, the compiler could, for example, implement an intrinsic, built-in function *_asm_builtin_atoi*(), which it recognized and expanded into inline assembly code. Then, in **<stdlib.h>**, there could be the following:

```
#define atoi(X) _asm_builtin_atoi(X)
```

The user's ``normal'' call to *atoi*() would then be expanded inline, but the implementor would also be required to provide a callable function named *atoi*() for use when the application requires it; for example, if its address is to be stored in a function pointer variable.

Implementors should note that since applications can **#undef** a macro in order to ensure that the function is used, this means that it is not safe for implementations to use the names of any standard functions in macro values, since the application could use **#undef** to ensure that no macro exists and then use the same name for an identifier with local scope. For example, historically it was common for a *getchar*() macro to be defined in **<stdio.h>** as:

```
#define getchar() getc(stdin)
```

This definition does not conform, because an application is allowed to use the identifier *getc* with local scope, and the expansion of the *getchar*() macro would then pick up the local *getc*. The following is conforming code, but would not compile with the above definition of *getchar*():

```
#include <stdio.h>
#undef getc

int main(void)
{
    int getc;

    getc = getchar();

    return getc;
}
```

This does not only affect function-like macros. For example, the following definition does not conform because there could be a local *sysconf* variable in scope when SIGRTMIN is expanded:

```
#define SIGRTMIN ((int)sysconf(_SC_SIGRT_MIN))
```

Implementors can avoid the problem by using aliases for standard functions instead of the

actual function, with names that conforming applications cannot use for local variables. For example:

```
#define SIGRTMIN ((int)__sysconf(_SC_SIGRT_MIN))
```

Austin Group Defect 655 is applied, making the requirement relating to explicit function declarations apply only to functions from the ISO C standard.

Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

Austin Group Defect 1404 is applied, adding to the examples of invalid values for function arguments.

B.2.1.2 Use and Implementation of Macros

There is no additional rationale provided for this section.

B.2.2 The Compilation Environment

B.2.2.1 POSIX.1 Symbols

This and the following section address the issue of ``name space pollution''. The ISO C standard requires that the name space beyond what it reserves not be altered except by explicit action of the application developer. This section defines the actions to add the POSIX.1 symbols for those headers where both the ISO C standard and POSIX.1 need to define symbols, and also where the XSI option extends the base standard.

When headers are used to provide symbols, there is a potential for introducing symbols that the application developer cannot predict. Ideally, each header should only contain one set of symbols, but this is not practical for historical reasons. Thus, the concept of feature test macros is included. Two feature test macros are explicitly defined by POSIX.1-2024; it is expected that future versions may add to this.

Note: Feature test macros allow an application to announce to the implementation its desire to have certain symbols and prototypes exposed. They should not be confused with the version test macros and constants for options in **<unistd.h>** which are the implementation's way of announcing functionality to the application.

It is further intended that these feature test macros apply only to the headers specified by POSIX.1-2024. Implementations are expressly permitted to make visible symbols not specified by POSIX.1-2024, within both POSIX.1 and other headers, under the control of feature test macros that are not defined by POSIX.1-2024.

The _POSIX_C_SOURCE Feature Test Macro

The POSIX.1-1990 standard specified a macro called _POSIX_SOURCE. This has been superseded by _POSIX_C_SOURCE. This symbol will allow implementations to support various versions of this standard simultaneously. For instance, when _POSIX_C_SOURCE is defined as 202405L, the system should make visible the same name space as permitted and required by the POSIX.1-2024 standard. A special case is the one where the implementation wishes to make available support for the 1990 version of the POSIX standard, in which instance when either _POSIX_SOURCE is defined or _POSIX_C_SOURCE is defined as 1, the system should make visible the same name space as permitted and required by the POSIX.1-1990 standard.

It is expected that C bindings to future POSIX standards will define new values for

128385 _POSIX_C_SOURCE, with each new value reserving the name space for that new standard.

128386 **The _XOPEN_SOURCE Feature Test Macro**

128387 The feature test macro _XOPEN_SOURCE is provided as the announcement mechanism for the
128388 application that it requires functionality from the Single UNIX Specification. _XOPEN_SOURCE
128389 must be defined to the value 800 before the inclusion of any header to enable the functionality in
128390 the Single UNIX Specification Version 5. Its definition subsumes the use of _POSIX_C_SOURCE.

128391 An extract of code from a conforming application, that appears before any **#include** statements,
128392 is given below:

128393 ```
#define _XOPEN_SOURCE 800 /* Single UNIX Specification, Version 5 */
```

128394 ```
#include ...
```

128395 Note that the definition of _XOPEN_SOURCE with the value 800 makes the definition of
128396 _POSIX_C_SOURCE redundant and it can safely be omitted.

128397 **The __STDC_WANT_LIB_EXT1__ Feature Test Macro**

128398 The ISO C standard specifies the feature test macro __STDC_WANT_LIB_EXT1__ as the
128399 announcement mechanism for the application that it requires functionality from Annex K. It
128400 specifies that the symbols specified in Annex K (if supported) are made visible when
128401 __STDC_WANT_LIB_EXT1__ is 1 and are not made visible when it is 0, but leaves it unspecified
128402 whether they are made visible when __STDC_WANT_LIB_EXT1__ is undefined. POSIX.1
128403 requires that they are not made visible when the macro is undefined (except for those symbols
128404 that are already explicitly allowed to be visible through the definition of _POSIX_C_SOURCE or
128405 _XOPEN_SOURCE, or both).

128406 POSIX.1 does not include the interfaces specified in Annex K of the ISO C standard, but allows
128407 the symbols to be made visible in headers when requested by the application in order that
128408 applications can use symbols from Annex K and symbols from POSIX.1 in the same translation
128409 unit.

128410 Austin Group Defect 1302 is applied, adding this subsection.

128411 *B.2.2.2 The Name Space*

128412 The reservation of identifiers is paraphrased from the ISO C standard. The text is included
128413 because it needs to be part of POSIX.1-2024, regardless of possible changes in future versions of
128414 the ISO C standard.

128415 These identifiers may be used by implementations, particularly for feature test macros.
128416 Implementations should not use feature test macro names that might be reasonably used by a
128417 standard.

128418 Including headers more than once is a reasonably common practice, and it should be carried
128419 forward from the ISO C standard. More significantly, having definitions in more than one
128420 header is explicitly permitted. Where the potential declaration is ``benign'' (the same definition
128421 twice) the declaration can be repeated, if that is permitted by the compiler. (This is usually true
128422 of macros, for example.) In those situations where a repetition is not benign (for example,
128423 **typedef**s), conditional compilation must be used. The situation actually occurs both within the
128424 ISO C standard and within POSIX.1: **time_t** should be in **<sys/types.h>**, and the ISO C standard
128425 mandates that it be in **<time.h>**.

128426 The area of name space pollution *versus* additions to structures is difficult because of the macro
128427 structure of C. The following discussion summarizes all the various problems with and

128428 objections to the issue.

128429 Note the phrase ``user-defined macro''. Users are not permitted to define macro names (or any
128430 other name) beginning with "_[A-Z_]". Thus, the conflict cannot occur for symbols reserved
128431 to the vendor's name space, and the permission to add fields automatically applies, without
128432 qualification, to those symbols.

128433 1. Data structures (and unions) need to be defined in headers by implementations to meet
128434 certain requirements of POSIX.1 and the ISO C standard.

128435 2. The structures defined by POSIX.1 are typically minimal, and any practical
128436 implementation would wish to add fields to these structures either to hold additional
128437 related information or for backwards-compatibility (or both). Future standards (and *de*
128438 *facto* standards) would also wish to add to these structures. Issues of field alignment
128439 make it impractical (at least in the general case) to simply omit fields when they are not
128440 defined by the particular standard involved.

128441 The **dirent** structure is an example of such a minimal structure (although one could argue
128442 about whether the other fields need visible names). The *st_rdev* field of most
128443 implementations' **stat** structure is a common example where extension is needed and
128444 where a conflict could occur.

128445 3. Fields in structures are in an independent name space, so the addition of such fields
128446 presents no problem to the C language itself in that such names cannot interact with
128447 identically named user symbols because access is qualified by the specific structure name.

128448 4. There is an exception to this: macro processing is done at a lexical level. Thus, symbols
128449 added to a structure might be recognized as user-provided macro names at the location
128450 where the structure is declared. This only can occur if the user-provided name is declared
128451 as a macro before the header declaring the structure is included. The user's use of the
128452 name after the declaration cannot interfere with the structure because the symbol is
128453 hidden and only accessible through access to the structure. Presumably, the user would
128454 not declare such a macro if there was an intention to use that field name.

128455 5. Macros from the same or a related header might use the additional fields in the structure,
128456 and those field names might also collide with user macros. Although this is a less
128457 frequent occurrence, since macros are expanded at the point of use, no constraint on the
128458 order of use of names can apply.

128459 6. An ``obvious'' solution of using names in the reserved name space and then redefining
128460 them as macros when they should be visible does not work because this has the effect of
128461 exporting the symbol into the general name space. For example, given a (hypothetical)
128462 system-provided header **<h.h>**, and two parts of a C program in **a.c** and **b.c**, in header
128463 **<h.h>**:

128464 ```
struct foo {
    int __i;
}
```

128467 ```
#ifdef _FEATURE_TEST
#define i __i;
#endif
```

128470 In file **a.c**:

128471 ```
#include h.h
extern int i;
...
```

  
```

In file **b.c**:

```
extern int i;
...
```

The symbol that the user thinks of as *i* in both files has an external name of *__i* in **a.c**; the same symbol *i* in **b.c** has an external name *i* (ignoring any hidden manipulations the compiler might perform on the names). This would cause a mysterious name resolution problem when **a.o** and **b.o** are linked.

Simply avoiding definition then causes alignment problems in the structure.

A structure of the form:

```
struct foo {
    union {
        int __i;
#ifdef _FEATURE_TEST
        int i;
#endif
    } __ii;
}
```

does not work because the name of the logical field *i* is *__ii.i*, and introduction of a macro to restore the logical name immediately reintroduces the problem discussed previously (although its manifestation might be more immediate because a syntax error would result if a recursive macro did not cause it to fail first).

7. A more workable solution would be to declare the structure:

```
struct foo {
#ifdef _FEATURE_TEST
    int i;
#else
    int __i;
#endif
}
```

However, if a macro (particularly one required by a standard) is to be defined that uses this field, two must be defined: one that uses *i*, the other that uses *__i*. If more than one additional field is used in a macro and they are conditional on distinct combinations of features, the complexity goes up as 2^n.

All this leaves a difficult situation: vendors must provide very complex headers to deal with what is conceptually simple and safe—adding a field to a structure. It is the possibility of user-provided macros with the same name that makes this difficult.

Several alternatives were proposed that involved constraining the user's access to part of the name space available to the user (as specified by the ISO C standard). In some cases, this was only until all the headers had been included. There were two proposals discussed that failed to achieve consensus:

1. Limiting it for the whole program.
2. Restricting the use of identifiers containing only uppercase letters until after all system headers had been included. It was also pointed out that because macros might wish to access fields of a structure (and macro expansion occurs totally at point of use) restricting names in this way would not protect the macro expansion, and thus the solution was inadequate.

It was finally decided that reservation of symbols would occur, but as constrained.

The current wording also allows the addition of fields to a structure, but requires that user macros of the same name not interfere. This allows vendors to do one of the following:

- Not create the situation (do not extend the structures with user-accessible names or use the solution in (7) above)
- Extend their compilers to allow some way of adding names to structures and macros safely

There are at least two ways that the compiler might be extended: add new preprocessor directives that turn off and on macro expansion for certain symbols (without changing the value of the macro) and a function or lexical operation that suppresses expansion of a word. The latter seems more flexible, particularly because it addresses the problem in macros as well as in declarations.

The following seems to be a possible implementation extension to the C language that will do this: any token that during macro expansion is found to be preceded by three `'#'` symbols shall not be further expanded in exactly the same way as described for macros that expand to their own name as in Section 6.10.3.4 of the ISO C standard. A vendor may also wish to implement this as an operation that is lexically a function, which might be implemented as:

```
#define __safe_name(x) ###x
```

Using a function notation would insulate vendors from changes in standards until such a functionality is standardized (if ever). Standardization of such a function would be valuable because it would then permit third parties to take advantage of it portably in software they may supply.

The symbols that are ``explicitly permitted, but not required by POSIX.1-2024'' include those classified below. (That is, the symbols classified below might, but are not required to, be present when _POSIX_C_SOURCE is defined to have the value 202405L.)

- Symbols in **<limits.h>** and **<unistd.h>** that are defined to indicate support for options or limits that are constant at compile-time
- Symbols in the name space reserved for the implementation by the ISO C standard
- Symbols in a name space reserved for a particular type of extension (for example, type names ending with **_t** in **<sys/types.h>**)
- Additional members of structures or unions whose names do not reduce the name space reserved for applications

Since both implementations and future versions of this standard and other POSIX standards may use symbols in the reserved spaces described in these tables, there is a potential for name space clashes. To avoid future name space clashes when adding symbols, implementations should not use the posix_, POSIX_, or _POSIX_ prefixes.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/2 is applied, deleting the entries POSIX_, _POSIX_, and posix_ from the column of allowed name space prefixes for use by an implementation in the first table. The presence of these prefixes was contradicting later text which states that: ``The prefixes posix_, POSIX_, and _POSIX are reserved for use by XCU Chapter 2 (on page 2472) and other POSIX standards. Implementations may add symbols to the headers shown in the following table, provided the identifiers ... do not use the reserved prefixes posix_, POSIX_, or _POSIX.''.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/3 is applied, correcting the reserved macro prefix from: ``PRI[a-z], SCN[a-z]'' to: ``PRI[Xa-z], SCN[Xa-z]'' in the second table. The change was needed since the ISO C standard allows implementations to define macros of the

form PRI or SCN followed by any lowercase letter or `'X'` in **<inttypes.h>**. (The ISO/IEC 9899:1999 standard, Subclause 7.26.4.)

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/4 is applied, adding a new section listing reserved names for the **<stdint.h>** header. This change is for alignment with the ISO C standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/2 is applied, making it clear that implementations are permitted to have symbols with the prefix _POSIX_ visible in any header.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/3 is applied, updating the table of allowed macro prefixes to include the prefix FP_[A-Z] for **<math.h>**. This text is added for consistency with the **<math.h>** reference page in the Base Definitions volume of POSIX.1-2024 which permits additional implementation-defined floating-point classifications.

Austin Group Interpretation 1003.1-2001 #048 is applied, reserving SEEK_ in the name space.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0001 [801], XSH/TC2-2008/0002 [780], XSH/TC2-2008/0003 [790], XSH/TC2-2008/0004 [780], XSH/TC2-2008/0005 [790], XSH/TC2-2008/0006 [782], XSH/TC2-2008/0007 [790], and XSH/TC2-2008/0008 [790] are applied.

Austin Group Defect 162 is applied, adding the **<endian.h>** header.

Austin Group Defect 697 is applied, reserving DT_ in the name space.

Austin Group Defect 845 is applied, reserving in6addr_ in the name space.

Austin Group Defect 993 is applied, reserving dli_ in the name space.

Austin Group Defect 1003 is applied, correcting a mismatch with the ISO C standard regarding reservation of each identifier with file scope described in the header section.

Austin Group Defect 1122 is applied, adding **<libintl.h>**.

Austin Group Defect 1151 is applied, adding ws_ as a reserved prefix for **<termios.h>**.

Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

Austin Group Defect 1456 is applied, clarifying the reservation of symbolic constants with the prefix _CS_, _PC_, and _SC_ for **<unistd.h>**.

B.2.3 Error Numbers

It was the consensus of the standard developers that to allow the conformance document to state that an error occurs and under what conditions, but to disallow a statement that it never occurs, does not make sense. It could be implied by the current wording that this is allowed, but to reduce the possibility of future interpretation requests, it is better to make an explicit statement.

The original ISO C standard just required that *errno* be a modifiable lvalue. Since the introduction of threads in 2011, the ISO C standard has instead required that *errno* be a macro which expands to a modifiable lvalue that has thread local storage duration.

Checking the value of *errno* alone is not sufficient to determine the existence or type of an error, since it is not required that a successful function call clear *errno*. The variable *errno* should only be examined when the return value of a function indicates that the value of *errno* is meaningful. In that case, the function is required to set the variable to something other than zero.

The variable *errno* is never set to zero by any function call; to do so would contradict the ISO C standard.

POSIX.1 requires (in the ERRORS sections of function descriptions) certain error values to be set

128606 in certain conditions because many existing applications depend on them. Some error numbers,
128607 such as [EFAULT], are entirely implementation-defined and are noted as such in their
128608 description in the ERRORS section. This section otherwise allows wide latitude to the
128609 implementation in handling error reporting.

128610 Some of the ERRORS sections in POSIX.1-2024 have two subsections. The first:

128611 ``The function shall fail if:''

128612 could be called the ``mandatory'' section.

128613 The second:

128614 ``The function may fail if:''

128615 could be informally known as the ``optional'' section.

128616 Attempting to infer the quality of an implementation based on whether it detects optional error
128617 conditions is not useful.

128618 Following each one-word symbolic name for an error, there is a description of the error. The
128619 rationale for some of the symbolic names follows:

128620 [ECANCELED] This spelling was chosen as being more common.

128621 [EFAULT] Most historical implementations do not catch an error and set *errno* when an
128622 invalid address is given to the functions *wait*( ), *time*( ), or *times*( ). Some
128623 implementations cannot reliably detect an invalid address. And most systems
128624 that detect invalid addresses will do so only for a system call, not for a library
128625 routine.

128626 [EFTYPE] This error code was proposed in earlier proposals as ``Inappropriate operation
128627 for file type'', meaning that the operation requested is not appropriate for the
128628 file specified in the function call. This code was proposed, although the same
128629 idea was covered by [ENOTTY], because the connotations of the name would
128630 be misleading. It was pointed out that the *fcntl*( ) function uses the error code
128631 [EINVAL] for this notion, and hence all instances of [EFTYPE] were changed
128632 to this code.

128633 [EINTR] POSIX.1 prohibits conforming implementations from restarting interrupted
128634 system calls of conforming applications unless the SA_RESTART flag is in
128635 effect for the signal. However, it does not require that [EINTR] be returned
128636 when another legitimate value may be substituted; for example, a partial
128637 transfer count when *read*( ) or *write*( ) are interrupted. This is only given when
128638 the signal-catching function returns normally as opposed to returns by
128639 mechanisms like *longjmp*( ) or *siglongjmp*( ).

128640 [ELOOP] In specifying conditions under which implementations would generate this
128641 error, the following goals were considered:

128642 • To ensure that actual loops are detected, including loops that result from
128643 symbolic links across distributed file systems.

128644 • To ensure that during pathname resolution an application can rely on
128645 the ability to follow at least {SYMLOOP_MAX} symbolic links in the
128646 absence of a loop.

128647 • To allow implementations to provide the capability of traversing more
128648 than {SYMLOOP_MAX} symbolic links in the absence of a loop.

- To allow implementations to detect loops and generate the error prior to encountering {SYMLOOP_MAX} symbolic links.

[ENAMETOOLONG]

When a symbolic link is encountered during pathname resolution, the contents of that symbolic link are used to create a new pathname. The standard developers intended to allow, but not require, that implementations enforce the restriction of {PATH_MAX} on the result of this pathname substitution.

Implementations are allowed, but not required, to treat a pathname longer than {PATH_MAX} passed into the system as an error. Implementations are required to return a pathname (even if it is longer than {PATH_MAX}) when the user supplies a buffer with an interface that specifies the buffer size, as long as the user-supplied buffer is large enough to hold the entire pathname (see XSH *getcwd*( ) for an example of this type of interface). Implementations are required to treat a request to pass a pathname longer than {PATH_MAX} from the system to a user-supplied buffer of an unspecified size (usually assumed to be of size {PATH_MAX}) as an error (see XSH *realpath*( ) for an example of this type of interface).

[ENOMEM] The term ``main memory'' is not used in POSIX.1 because it is implementation-defined.

[ENOTSUP] This error code is to be used when an implementation chooses to implement the required functionality of POSIX.1-2024 but does not support optional facilities defined by POSIX.1-2024. In some earlier versions of this standard, the difference between [ENOTSUP] and [ENOSYS] was that [ENOSYS] indicated that the function was not supported at all. This is no longer the case as [ENOSYS] can also be used to indicate non-support of optional functionality for a function that has some required functionality. (See XSH *encrypt*( ).)

[ENOTTY] The symbolic name for this error is derived from a time when device control was done by *ioctl*( ) and that operation was only permitted on a terminal interface. The term ``TTY'' is derived from ``teletypewriter'', the devices to which this error originally applied.

[EOVERFLOW] Most of the uses of this error code are related to large file support. Typically, these cases occur on systems which support multiple programming environments with different sizes for **off_t**, but they may also occur in connection with remote file systems.

In addition, when different programming environments have different widths for types such as **int** and **uid_t**, several functions may encounter a condition where a value in a particular environment is too wide to be represented. In that case, this error should be raised. For example, suppose the currently running process has 64-bit **int**, and file descriptor 9 223 372 036 854 775 807 is open and does not have the close-on-*exec* flag set. If the process then uses *execl*( ) to *exec* a file compiled in a programming environment with 32-bit **int**, the call to *execl*( ) can fail with *errno* set to [EOVERFLOW]. A similar failure can occur with *execl*( ) if any of the user IDs or any of the group IDs to be assigned to the new process image are out of range for the executed file's programming environment.

Note, however, that this condition cannot occur for functions that are explicitly described as always being successful, such as *getpid*( ).

[EPIPE] This condition normally generates the signal SIGPIPE; the error is returned if the generation of the signal is suppressed or the signal does not terminate the process.

[EROFS] In historical implementations, attempting to *unlink*( ) or *rmdir*( ) a mount point would generate an [EBUSY] error. An implementation could be envisioned where such an operation could be performed without error. In this case, if *either* the directory entry or the actual data structures reside on a read-only file system, [EROFS] is the appropriate error to generate. (For example, changing the link count of a file on a read-only file system could not be done, as is required by *unlink*( ), and thus an error should be reported.)

Three error numbers, [EDOM], [EILSEQ], and [ERANGE], were added to this section primarily for consistency with the ISO C standard.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0009 [496] and XSH/TC2-2008/0010 [681] are applied.

Austin Group Defect 1067 is applied, adding [ESOCKTNOSUPPORT].

Austin Group Defect 1380 is applied, changing the descriptions of [EMLINK] and [EXDEV].

Austin Group Defect 1669 is applied, changing the description of [EFBIG].

Alternative Solutions for Per-Thread errno

The historical implementation of *errno* as a single global variable does not work in a multi-threaded environment. In such an environment, a thread may make a POSIX.1 call and get a –1 error return, but before that thread can check the value of *errno*, another thread might have made a second POSIX.1 call that also set *errno*. This behavior is unacceptable in robust programs. There were a number of alternatives that were considered for handling the *errno* problem:

- Implement *errno* as a per-thread integer variable.
- Implement *errno* as a service that can access the per-thread error number.
- Change all POSIX.1 calls to accept an extra status argument and avoid setting *errno*.
- Change all POSIX.1 calls to raise a language exception.

The first option offers the highest level of compatibility with existing practice but requires special support in the linker, compiler, and/or virtual memory system to support the new concept of thread private variables. When compared with current practice, the third and fourth options are much cleaner, more efficient, and encourage a more robust programming style, but they require new versions of all of the POSIX.1 functions that might detect an error. The second option offers compatibility with existing code that uses the **<errno.h>** header to define the symbol *errno*. In this option, *errno* may be a macro defined:

```
#define errno  (*__errno())
extern int      *__errno();
```

This option may be implemented as a per-thread variable whereby an *errno* field is allocated in the user space object representing a thread, and whereby the function *__errno*( ) makes a system call to determine the location of its user space object and returns the address of the *errno* field of that object. Another implementation, one that avoids calling the kernel, involves allocating stacks in chunks. The stack allocator keeps a side table indexed by chunk number containing a pointer to the thread object that uses that chunk. The *__errno*( ) function then looks at the stack pointer, determines the chunk number, and uses that as an index into the chunk table to find its thread object and thus its private value of *errno*. On most architectures, this can be done in four

to five instructions. Some compilers may wish to implement *__errno*() inline to improve performance.

Disallowing Return of the [EINTR] Error Code

Many blocking interfaces defined by POSIX.1-2024 may return [EINTR] if interrupted during their execution by a signal handler. Blocking interfaces introduced under the threads functionality do not have this property. Instead, they require that the interface appear to be atomic with respect to interruption. In particular, applications calling blocking interfaces need not handle any possible [EINTR] return as a special case since it will never occur. In the case of threads functions in **<threads.h>**, the requirement is stated in terms of the call not being affected if the calling thread executes a signal handler during the call, since these functions return errors in a different way and cannot distinguish an [EINTR] condition from other error conditions. If it is necessary to restart operations or complete incomplete operations following the execution of a signal handler, this is handled by the implementation, rather than by the application.

Requiring applications to handle [EINTR] errors on blocking interfaces has been shown to be a frequent source of often unreproducible bugs, and it adds no compelling value to the available functionality. Thus, blocking interfaces introduced for use by multi-threaded programs do not use this paradigm. In particular, in none of the functions *flockfile*(), *pthread_cond_timedwait*(), *pthread_cond_wait*(), *pthread_join*(), *pthread_mutex_lock*(), and *sigwait*() did providing [EINTR] returns add value, or even particularly make sense. Thus, these functions do not provide for an [EINTR] return, even when interrupted by a signal handler. The same arguments can be applied to *sem_wait*(), *sem_trywait*(), *sigwaitinfo*(), and *sigtimedwait*(), but implementations are permitted to return [EINTR] error codes for these functions for compatibility with earlier versions of this standard. Applications cannot rely on calls to these functions returning [EINTR] error codes when signals are delivered to the calling thread, but they should allow for the possibility.

Austin Group Interpretation 1003.1-2001 #050 is applied, allowing [ENOTSUP] and [EOPNOTSUPP] to be the same values.

B.2.3.1 Additional Error Numbers

The ISO C standard defines the name space for implementations to add additional error numbers.

B.2.4 Signal Concepts

Historical implementations of signals, using the *signal*() function, have shortcomings that make them unreliable for many application uses. Because of this, a new signal mechanism, based very closely on the one of 4.2 BSD and 4.3 BSD, was added to POSIX.1.

Signal Names

The restriction on the actual type used for **sigset_t** is intended to guarantee that these objects can always be assigned, have their address taken, and be passed as parameters by value. It is not intended that this type be a structure including pointers to other data structures, as that could impact the portability of applications performing such operations. A reasonable implementation could be a structure containing an array of some integer type.

The signals described in POSIX.1-2024 must have unique values so that they may be named as parameters of **case** statements in the body of a C-language **switch** clause. However, implementation-defined signals may have values that overlap with each other or with signals

specified in POSIX.1-2024. An example of this is SIGABRT, which traditionally overlaps some other signal, such as SIGIOT.

SIGKILL, SIGTERM, SIGUSR1, and SIGUSR2 are ordinarily generated only through the explicit use of the *kill*() function, although some implementations generate SIGKILL under extraordinary circumstances. SIGTERM is traditionally the default signal sent by the *kill* command.

The signals SIGBUS, SIGEMT, SIGIOT, SIGTRAP, and SIGSYS were omitted from POSIX.1 because their behavior is implementation-defined and could not be adequately categorized. Conforming implementations may deliver these signals, but must document the circumstances under which they are delivered and note any restrictions concerning their delivery. The signals SIGFPE, SIGILL, and SIGSEGV are similar in that they also generally result only from programming errors. They were included in POSIX.1 because they do indicate three relatively well-categorized conditions. They are all defined by the ISO C standard and thus would have to be defined by any system with an ISO C standard binding, even if not explicitly included in POSIX.1.

There is very little that a Conforming POSIX.1 Application can do by catching, ignoring, or masking any of the signals SIGILL, SIGTRAP, SIGIOT, SIGEMT, SIGBUS, SIGSEGV, SIGSYS, or SIGFPE. They will generally be generated by the system only in cases of programming errors. While it may be desirable for some robust code (for example, a library routine) to be able to detect and recover from programming errors in other code, these signals are not nearly sufficient for that purpose. One portable use that does exist for these signals is that a command interpreter can recognize them as the cause of termination of a process (with *wait*()) and print an appropriate message. The mnemonic tags for these signals are derived from their PDP-11 origin.

The signals SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU, and SIGCONT are provided for job control and are unchanged from 4.2 BSD. The signal SIGCHLD is also typically used by job control shells to detect children that have terminated or, as in 4.2 BSD, stopped.

Some implementations, including System V, have a signal named SIGCLD, which is similar to SIGCHLD in 4.2 BSD. POSIX.1 permits implementations to have a single signal with both names. POSIX.1 carefully specifies ways in which conforming applications can avoid the semantic differences between the two different implementations. The name SIGCHLD was chosen for POSIX.1 because most current application usages of it can remain unchanged in conforming applications. SIGCLD in System V has more cases of semantics that POSIX.1 does not specify, and thus applications using it are more likely to require changes in addition to the name change.

The signals SIGUSR1 and SIGUSR2 are commonly used by applications for notification of exceptional behavior and are described as ``reserved as application-defined'' so that such use is not prohibited. Implementations should not generate SIGUSR1 or SIGUSR2, except when explicitly requested by *kill*(). It is recommended that libraries not use these two signals, as such use in libraries could interfere with their use by applications calling the libraries. If such use is unavoidable, it should be documented. It is prudent for non-portable libraries to use non-standard signals to avoid conflicts with use of standard signals by portable libraries.

There is no portable way for an application to catch or ignore non-standard signals. Some implementations define the range of signal numbers, so applications can install signal-catching functions for all of them. Unfortunately, implementation-defined signals often cause problems when caught or ignored by applications that do not understand the reason for the signal. While the desire exists for an application to be more robust by handling all possible signals (even those only generated by *kill*()), no existing mechanism was found to be sufficiently portable to include in POSIX.1. The value of such a mechanism, if included, would be diminished given that SIGKILL would still not be catchable.

A number of new signal numbers are reserved for applications because the two user signals defined by POSIX.1 are insufficient for many realtime applications. A range of signal numbers is specified, rather than an enumeration of additional reserved signal names, because different applications and application profiles will require a different number of application signals. It is not desirable to burden all application domains and therefore all implementations with the maximum number of signals required by all possible applications. Note that in this context, signal numbers are essentially different signal priorities.

The relatively small number of required additional signals, {_POSIX_RTSIG_MAX}, was chosen so as not to require an unreasonably large signal mask/set. While this number of signals defined in POSIX.1 will fit in a single 32-bit word signal mask, it is recognized that most existing implementations define many more signals than are specified in POSIX.1 and, in fact, many implementations have already exceeded 32 signals (including the ``null signal''). Support of {_POSIX_RTSIG_MAX} additional signals may push some implementation over the single 32-bit word line, but is unlikely to push any implementations that are already over that line beyond the 64-signal line.

B.2.4.1 *Signal Generation and Delivery*

The terms defined in this section are not used consistently in documentation of historical systems. Each signal can be considered to have a lifetime beginning with generation and ending with delivery or acceptance. The POSIX.1 definition of ``delivery'' does not exclude ignored signals; this is considered a more consistent definition. This revised text in several parts of POSIX.1-2024 clarifies the distinct semantics of asynchronous signal delivery and synchronous signal acceptance. The previous wording attempted to categorize both under the term ``delivery'', which led to conflicts over whether the effects of asynchronous signal delivery applied to synchronous signal acceptance.

Signals generated for a process are delivered to only one thread. Thus, if more than one thread is eligible to receive a signal, one has to be chosen. The choice of threads is left entirely up to the implementation both to allow the widest possible range of conforming implementations and to give implementations the freedom to deliver the signal to the ``easiest possible'' thread should there be differences in ease of delivery between different threads.

Note that should multiple delivery among cooperating threads be required by an application, this can be trivially constructed out of the provided single-delivery semantics. The construction of a *sigwait_multiple*() function that accomplishes this goal is presented with the rationale for *sigwaitinfo*().

Implementations should deliver unblocked signals as soon after they are generated as possible. However, it is difficult for POSIX.1 to make specific requirements about this, beyond those in *kill*() and *sigprocmask*(). Even on systems with prompt delivery, scheduling of higher priority processes is always likely to cause delays.

In general, the interval between the generation and delivery of unblocked signals cannot be detected by an application. Thus, references to pending signals generally apply to blocked, pending signals. An implementation registers a signal as pending on the process when no thread has the signal unblocked and there are no threads blocked in a *sigwait*() function for that signal. Thereafter, the implementation delivers the signal to the first thread that unblocks the signal or calls a *sigwait*() function on a signal set containing this signal rather than choosing the recipient thread at the time the signal is sent.

In the 4.3 BSD system, signals that are blocked and set to SIG_IGN are discarded immediately upon generation. For a signal that is ignored as its default action, if the action is SIG_DFL and the signal is blocked, a generated signal remains pending. In the 4.1 BSD system and in System V Release 3 (two other implementations that support a somewhat similar signal

mechanism), all ignored blocked signals remain pending if generated. Because it is not normally useful for an application to simultaneously ignore and block the same signal, it was unnecessary for POSIX.1 to specify behavior that would invalidate any of the historical implementations.

There is one case in some historical implementations where an unblocked, pending signal does not remain pending until it is delivered. In the System V implementation of *signal*( ), pending signals are discarded when the action is set to SIG_DFL or a signal-catching routine (as well as to SIG_IGN). Except in the case of setting SIGCHLD to SIG_DFL, implementations that do this do not conform completely to POSIX.1. Some earlier proposals for POSIX.1 explicitly stated this, but these statements were redundant due to the requirement that functions defined by POSIX.1 not change attributes of processes defined by POSIX.1 except as explicitly stated.

POSIX.1 specifically states that the order in which multiple, simultaneously pending signals are delivered is unspecified. This order has not been explicitly specified in historical implementations, but has remained quite consistent and been known to those familiar with the implementations. Thus, there have been cases where applications (usually system utilities) have been written with explicit or implicit dependencies on this order. Implementors and others porting existing applications may need to be aware of such dependencies.

When there are multiple pending signals that are not blocked, implementations should arrange for the delivery of all signals at once, if possible. Some implementations stack calls to all pending signal-catching routines, making it appear that each signal-catcher was interrupted by the next signal. In this case, the implementation should ensure that this stacking of signals does not violate the semantics of the signal masks established by *sigaction*( ). Other implementations process at most one signal when the operating system is entered, with remaining signals saved for later delivery. Although this practice is widespread, this behavior is neither standardized nor endorsed. In either case, implementations should attempt to deliver signals associated with the current state of the process (for example, SIGFPE) before other signals, if possible.

In 4.2 BSD and 4.3 BSD, it is not permissible to ignore or explicitly block SIGCONT, because if blocking or ignoring this signal prevented it from continuing a stopped process, such a process could never be continued (only killed by SIGKILL). However, 4.2 BSD and 4.3 BSD do block SIGCONT during execution of its signal-catching function when it is caught, creating exactly this problem. A proposal was considered to disallow catching SIGCONT in addition to ignoring and blocking it, but this limitation led to objections. The consensus was to require that SIGCONT always continue a stopped process when generated. This removed the need to disallow ignoring or explicit blocking of the signal; note that SIG_IGN and SIG_DFL are equivalent for SIGCONT.

B.2.4.2 *Realtime Signal Generation and Delivery*

The realtime signals functionality is required in this version of the standard for the following reasons:

- The **sigevent** structure is used by other POSIX.1 functions that result in asynchronous event notifications to specify the notification mechanism to use and other information needed by the notification mechanism. POSIX.1-2024 defines only three symbolic values for the notification mechanism:
  - — SIGEV_NONE is used to indicate that no notification is required when the event occurs. This is useful for applications that use asynchronous I/O with polling for completion.
  - — SIGEV_SIGNAL indicates that a signal is generated when the event occurs.

128928 — SIGEV_THREAD provides for ``callback functions'' for asynchronous notifications
128929 done by a function call within the context of a new thread. This provides a multi-
128930 threaded process with a more natural means of notification than signals.

128931 The primary difficulty with previous notification approaches has been to specify the
128932 environment of the notification routine.

128933 — One approach is to limit the notification routine to call only functions permitted in a
128934 signal handler. While the list of permissible functions is clearly stated, this is overly
128935 restrictive.

128936 — A second approach is to define a new list of functions or classes of functions that are
128937 explicitly permitted or not permitted. This would give a programmer more lists to
128938 deal with, which would be awkward.

128939 — The third approach is to define completely the environment for execution of the
128940 notification function. A clear definition of an execution environment for notification
128941 is provided by executing the notification function in the environment of a newly
128942 created thread.

128943 Implementations may support additional notification mechanisms by defining new values
128944 for *sigev_notify*.

128945 For a notification type of SIGEV_SIGNAL, the other members of the **sigevent** structure
128946 defined by POSIX.1-2024 specify the realtime signal—that is, the signal number and
128947 application-defined value that differentiates between occurrences of signals with the same
128948 number—that will be generated when the event occurs. The structure is defined in
128949 **<signal.h>**, even though the structure is not directly used by any of the signal functions,
128950 because it is part of the signals interface used by the POSIX.1b ``client functions''. When the
128951 client functions include **<signal.h>** to define the signal names, the **sigevent** structure will
128952 also be defined.

128953 An application-defined value passed to the signal handler is used to differentiate between
128954 different ``events'' instead of requiring that the application use different signal numbers for
128955 several reasons:

128956 — Realtime applications potentially handle a very large number of different events.
128957 Requiring that implementations support a correspondingly large number of distinct
128958 signal numbers will adversely impact the performance of signal delivery because the
128959 signal masks to be manipulated on entry and exit to the handlers will become large.

128960 — Event notifications are prioritized by signal number (the rationale for this is
128961 explained in the following paragraphs) and the use of different signal numbers to
128962 differentiate between the different event notifications overloads the signal number
128963 more than has already been done. It also requires that the application developer
128964 make arbitrary assignments of priority to events that are logically of equal priority.

128965 A union is defined for the application-defined value so that either an integer constant or a
128966 pointer can be portably passed to the signal-catching function. On some architectures a
128967 pointer cannot be cast to an **int** and *vice versa*.

128968 Use of a structure here with an explicit notification type discriminant rather than explicit
128969 parameters to realtime functions, or embedded in other realtime structures, provides for
128970 future extensions to POSIX.1-2024. Additional, perhaps more efficient, notification
128971 mechanisms can be supported for existing realtime function interfaces, such as timers and
128972 asynchronous I/O, by extending the **sigevent** structure appropriately. The existing
128973 realtime function interfaces will not have to be modified to use any such new notification
128974 mechanism. The revised text concerning the SIGEV_SIGNAL value makes consistent the

semantics of the members of the **sigevent** structure, particularly in the definitions of *lio_listio*() and *aio_fsync*(). For uniformity, other revisions cause this specification to be referred to rather than inaccurately duplicated in the descriptions of functions and structures using the **sigevent** structure. The revised wording does not relax the requirement that the signal number be in the range SIGRTMIN to SIGRTMAX to guarantee queuing and passing of the application value, since that requirement is still implied by the signal names.

- POSIX.1-2024 is intentionally vague on whether ``non-realtime'' signal-generating mechanisms can result in a **siginfo_t** being supplied to the handler on delivery. In one existing implementation, a **siginfo_t** is posted on signal generation, even though the implementation does not support queuing of multiple occurrences of a signal. It is not the intent of POSIX.1-2024 to preclude this, independent of the mandate to define signals that do support queuing. Any interpretation that appears to preclude this is a mistake in the reading or writing of the standard.

- Signals handled by realtime signal handlers might be generated by functions or conditions that do not allow the specification of an application-defined value and do not queue. POSIX.1-2024 specifies the *si_code* member of the **siginfo_t** structure used in existing practice and defines additional codes so that applications can detect whether an application-defined value is present or not. The code SI_USER for *kill*()-generated signals is adopted from existing practice.

- The *sigaction*() *sa_flags* value SA_SIGINFO tells the implementation that the signal-catching function expects two additional arguments. When the flag is not set, a single argument, the signal number, is passed as specified by POSIX.1-2024. Although POSIX.1-2024 does not explicitly allow the *info* argument to the handler function to be NULL, this is existing practice. This provides for compatibility with programs whose signal-catching functions are not prepared to accept the additional arguments. POSIX.1-2024 is explicitly unspecified as to whether signals actually queue when SA_SIGINFO is not set for a signal, as there appear to be no benefits to applications in specifying one behavior or another. One existing implementation queues a **siginfo_t** on each signal generation, unless the signal is already pending, in which case the implementation discards the new **siginfo_t**; that is, the queue length is never greater than one. This implementation only examines SA_SIGINFO on signal delivery, discarding the queued **siginfo_t** if its delivery was not requested.

  The third argument to the signal-catching function, *context*, is left undefined by POSIX.1-2024, but is specified in the interface because it matches existing practice for the SA_SIGINFO flag. It was considered undesirable to require a separate implementation for SA_SIGINFO for POSIX conformance on implementations that already support the two additional parameters.

- The requirement to deliver lower numbered signals in the range SIGRTMIN to SIGRTMAX first, when multiple unblocked signals are pending, results from several considerations:

  — A method is required to prioritize event notifications. The signal number was chosen instead of, for instance, associating a separate priority with each request, because an implementation has to check pending signals at various points and select one for delivery when more than one is pending. Specifying a selection order is the minimal additional semantic that will achieve prioritized delivery. If a separate priority were to be associated with queued signals, it would be necessary for an implementation to search all non-empty, non-blocked signal queues and select from among them the pending signal with the highest priority. This would significantly increase the cost of and decrease the determinism of signal delivery.

— Given the specified selection of the lowest numeric unblocked pending signal, preemptive priority signal delivery can be achieved using signal numbers and signal masks by ensuring that the *sa_mask* for each signal number blocks all signals with a higher numeric value.

For realtime applications that want to use only the newly defined realtime signal numbers without interference from the standard signals, this can be achieved by blocking all of the standard signals in the thread signal mask and in the *sa_mask* installed by the signal action for the realtime signal handlers.

POSIX.1-2024 explicitly leaves unspecified the ordering of signals outside of the range of realtime signals and the ordering of signals within this range with respect to those outside the range. It was believed that this would unduly constrain implementations or standards in the future definition of new signals.

Austin Group Defect 633 is applied, reducing to two the allowed behaviors for the signal mask of the thread that is created to handle a SIGEV_THREAD notification.

Austin Group Defect 1116 is applied, removing a reference to the Realtime Signals Extension option that existed in earlier versions of this standard.

B.2.4.3 Signal Actions

Early proposals mentioned SIGCONT as a second exception to the rule that signals are not delivered to stopped processes until continued. Because POSIX.1-2024 now specifies that SIGCONT causes the stopped process to continue when it is generated, delivery of SIGCONT is not prevented because a process is stopped, even without an explicit exception to this rule.

Ignoring a signal by setting the action to SIG_IGN (or SIG_DFL for signals whose default action is to ignore) is not the same as installing a signal-catching function that simply returns. Invoking such a function will interrupt certain system functions that block processes (for example, *wait*( ), *sigsuspend*( ), *pause*( ), *read*( ), *write*( )) while ignoring a signal has no such effect on the process.

Historical implementations discard pending signals when the action is set to SIG_IGN. However, they do not always do the same when the action is set to SIG_DFL and the default action is to ignore the signal. POSIX.1-2024 requires this for the sake of consistency and also for completeness, since the only signal this applies to is SIGCHLD, and POSIX.1-2024 disallows setting its action to SIG_IGN.

Some implementations (System V, for example) assign different semantics for SIGCLD depending on whether the action is set to SIG_IGN or SIG_DFL. Since POSIX.1 requires that the default action for SIGCHLD be to ignore the signal, applications should always set the action to SIG_DFL in order to avoid SIGCHLD.

Whether or not an implementation allows SIG_IGN as a SIGCHLD disposition to be inherited across a call to one of the *exec* family of functions or *posix_spawn*( ) is explicitly left as unspecified. This change was made as a result of IEEE PASC Interpretation 1003.1 #132, and permits the implementation to decide between the following alternatives:

- Unconditionally leave SIGCHLD set to SIG_IGN, in which case the implementation would not allow applications that assume inheritance of SIG_DFL to conform to POSIX.1-2024 without change. The implementation would, however, retain an ability to control applications that create child processes but never call on the *wait* family of functions, potentially filling up the process table.
- Unconditionally reset SIGCHLD to SIG_DFL, in which case the implementation would allow applications that assume inheritance of SIG_DFL to conform. The implementation would, however, lose an ability to control applications that spawn child processes but

never reap them.

- Provide some mechanism, not specified in POSIX.1-2024, to control inherited SIGCHLD dispositions.

Some implementations (System V, for example) will deliver a SIGCLD signal immediately when a process establishes a signal-catching function for SIGCLD when that process has a child that has already terminated. Other implementations, such as 4.3 BSD, do not generate a new SIGCHLD signal in this way. In general, a process should not attempt to alter the signal action for the SIGCHLD signal while it has any outstanding children. However, it is not always possible for a process to avoid this; for example, shells sometimes start up processes in pipelines with other processes from the pipeline as children. Processes that cannot ensure that they have no children when altering the signal action for SIGCHLD thus need to be prepared for, but not depend on, generation of an immediate SIGCHLD signal.

The default action of the stop signals (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is to stop a process that is executing. If a stop signal is delivered to a process that is already stopped, it has no effect. In fact, if a stop signal is generated for a stopped process whose signal mask blocks the signal, the signal will never be delivered to the process since the process must receive a SIGCONT, which discards all pending stop signals, in order to continue executing.

The SIGCONT signal continues a stopped process even if SIGCONT is blocked (or ignored). However, if a signal-catching routine has been established for SIGCONT, it will not be entered until SIGCONT is unblocked.

If a process in an orphaned process group stops, it is no longer under the control of a job control shell and hence would not normally ever be continued. Because of this, orphaned processes that receive terminal-related stop signals (SIGTSTP, SIGTTIN, SIGTTOU, but not SIGSTOP) must not be allowed to stop. The goal is to prevent stopped processes from languishing forever. (As SIGSTOP is sent only via *kill*(), it is assumed that the process or user sending a SIGSTOP can send a SIGCONT when desired.) Instead, the system must discard the stop signal. As an extension, it may also deliver another signal in its place. 4.3 BSD sends a SIGKILL, which is overly effective because SIGKILL is not catchable. Another possible choice is SIGHUP. 4.3 BSD also does this for orphaned processes (processes whose parent has terminated) rather than for members of orphaned process groups; this is less desirable because job control shells manage process groups. POSIX.1 also prevents SIGTTIN and SIGTTOU signals from being generated for processes in orphaned process groups as a direct result of activity on a terminal, preventing infinite loops when *read*() and *write*() calls generate signals that are discarded; see Section A.11.1.4 (on page 3720). A similar restriction on the generation of SIGTSTP was considered, but that would be unnecessary and more difficult to implement due to its asynchronous nature.

Although POSIX.1 requires that signal-catching functions be called with only one argument, there is nothing to prevent conforming implementations from extending POSIX.1 to pass additional arguments, as long as Strictly Conforming POSIX.1 Applications continue to compile and execute correctly. Most historical implementations do, in fact, pass additional, signal-specific arguments to certain signal-catching routines.

There was a proposal to change the declared type of the signal handler to:

```
void func (int sig, ...);
```

The usage of ellipses (`"..."`) is ISO C standard syntax to indicate a variable number of arguments. Its use was intended to allow the implementation to pass additional information to the signal handler in a standard manner.

Unfortunately, this construct would require all signal handlers to be defined with this syntax because the ISO C standard allows implementations to use a different parameter passing mechanism for variable parameter lists than for non-variable parameter lists. Thus, all existing

signal handlers in all existing applications would have to be changed to use the variable syntax in order to be standard and portable. This is in conflict with the goal of Minimal Changes to Existing Application Code.

When terminating a process from a signal-catching function, processes should be aware of any interpretation that their parent may make of the status returned by *wait*(), *waitid*(), or *waitpid*(). In particular, a signal-catching function should not call *exit*(0) or *_exit*(0) unless it wants to indicate successful termination. A non-zero argument to *exit*() or *_exit*() can be used to indicate unsuccessful termination. Alternatively, the process can use *kill*() to send itself a fatal signal (first ensuring that the signal is set to the default action and not blocked). See also the RATIONALE section of the *_exit*() function.

The behavior of *unsafe* functions, as defined by this section, is undefined when they are called from (or after a *longjmp*() or *siglongjmp*() out of) signal-catching functions in certain circumstances. The behavior of async-signal-safe functions, as defined by this section, is as specified by POSIX.1, regardless of invocation from a signal-catching function. This is the only intended meaning of the statement that async-signal-safe functions may be used in signal-catching functions without restriction. Applications must still consider all effects of such functions on such things as data structures, files, and process state. In particular, application developers need to consider the restrictions on interactions when interrupting *sleep*() (see *sleep*()) and interactions among multiple handles for a file description. The fact that any specific function is listed as async-signal-safe does not necessarily mean that invocation of that function from a signal-catching function is recommended.

In order to prevent errors arising from interrupting non-async-signal-safe function calls, applications should protect calls to these functions either by blocking the appropriate signals or through the use of some programmatic semaphore. POSIX.1 does not address the more general problem of synchronizing access to shared data structures. Note in particular that even the ``safe'' functions may modify the global variable *errno*; the signal-catching function may want to save and restore its value. The same principles apply to the async-signal-safety of application routines and asynchronous data access.

Note that although *longjmp*() and *siglongjmp*() are in the list of async-signal-safe functions, there are restrictions on subsequent behavior after the function is called from a signal-catching function. This is because the code executing after *longjmp*() or *siglongjmp*() can call any unsafe functions with the same danger as calling those unsafe functions directly from the signal handler. Applications that use *longjmp*() or *siglongjmp*() out of signal handlers require rigorous protection in order to be portable. Many of the other functions that are excluded from the list are traditionally implemented using either the C language *malloc*() or *free*() functions or the ISO C standard I/O library, both of which traditionally use data structures in a non-async-signal-safe manner. Because any combination of different functions using a common data structure can cause async-signal-safety problems, POSIX.1 does not define the behavior when any unsafe function is called in (or after a *longjmp*() or *siglongjmp*() out of) a signal handler that interrupts any unsafe function or the non-async-signal-safe processing equivalent to *exit*() that is performed after return from the initial call to *main*().

The only realtime extension to signal actions is the addition of the additional parameters to the signal-catching function. This extension has been explained and motivated in the previous section. In making this extension, though, developers of POSIX.1b ran into issues relating to function prototypes. In response to input from the POSIX.1 standard developers, members were added to the **sigaction** structure to specify function prototypes for the newer signal-catching function specified by POSIX.1b. These members follow changes that are being made to POSIX.1. Note that POSIX.1-2024 explicitly states that these fields may overlap so that a union can be defined. This enabled existing implementations of POSIX.1 to maintain binary-compatibility when these extensions were added.

The **siginfo_t** structure was adopted for passing the application-defined value to match existing practice, but the existing practice has no provision for an application-defined value, so this was added. Note that POSIX normally reserves the ``_t'' type designation for opaque types. The **siginfo_t** structure breaks with this convention to follow existing practice and thus promote portability.

POSIX.1-2024 specifies several values for the *si_code* member of the **siginfo_t** structure. Some were introduced in POSIX.1b; others were XSI functionality in the Single UNIX Specification, Version 2 and Version 3, that has now become Base functionality. Historically, an *si_code* value of less than or equal to zero indicated that the signal was generated by a process via the *kill*() function, and values of *si_code* that provided additional information for implementation-generated signals, such as SIGFPE or SIGSEGV, were all positive. This functionality is partially specified for XSI systems in that if *si_code* is less than or equal to zero, the signal was generated by a process. However, since POSIX.1b did not specify that SI_USER (or SI_QUEUE) had a value less than or equal to zero, it is not true that when the signal is generated by a process, the value of *si_code* will always be less than or equal to zero. XSI applications should check whether *si_code* is SI_USER or SI_QUEUE in addition to checking whether it is less than or equal to zero. Applications on systems that do not support the XSI option should just check for SI_USER and SI_QUEUE.

If an implementation chooses to define additional values for *si_code*, these values have to be different from the values of the non-signal-specific symbols specified by POSIX.1-2024. This will allow conforming applications to differentiate between signals generated by standard events and those generated by other implementation events in a manner compatible with existing practice.

The unique values of *si_code* for the POSIX.1b asynchronous events have implications for implementations of, for example, asynchronous I/O or message passing in user space library code. Such an implementation will be required to provide a hidden interface to the signal generation mechanism that allows the library to specify the standard values of *si_code*.

POSIX.1-2024 also specifies additional members of **siginfo_t**, beyond those that were in POSIX.1b. Like the *si_code* values mentioned above, these were XSI functionality in the Single UNIX Specification, Version 2 and Version 3, that has now become Base functionality. They provide additional information when *si_code* has one of the values that moved from XSI to Base.

Although it is not explicitly visible to applications, there are additional semantics for signal actions implied by queued signals and their interaction with other POSIX.1b realtime functions. Specifically:

- It is not necessary to queue signals whose action is SIG_IGN.
- For implementations that support POSIX.1b timers, some interaction with the timer functions at signal delivery is implied to manage the timer overrun count.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/5 is applied, reordering the RTS shaded text under the third and fourth paragraphs of the SIG_DFL description. This corrects an earlier editorial error in this section.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/6 is applied, adding the *abort*() function to the list of async-signal-safe functions.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/4 is applied, adding the *sockatmark*() function to the list of async-signal-safe functions.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0011 [690], XSH/TC2-2008/0012 [516], XSH/TC2-2008/0013 [692], XSH/TC2-2008/0014 [615], XSH/TC2-2008/0015 [516], and XSH/TC2-2008/0016 [807] are applied.

Austin Group Defect 62 is applied, adding the *_Fork*() function to, and removing the *fork*() function from, the list of async-signal-safe functions.

Austin Group Defect 162 is applied, adding functions from the **<endian.h>** header to the list of async-signal-safe functions.

Austin Group Defect 411 is applied, adding *accept4*(), *dup3*(), and *pipe2*() to the list of async-signal-safe functions.

Austin Group Defect 614 is applied, adding *posix_close*() to the list of async-signal-safe functions.

Austin Group Defect 699 is applied, adding *setegid*(), *seteuid*(), *setregid*(), and *setreuid*() to the list of async-signal-safe functions.

Austin Group Defect 711 is applied, adding *va_arg*(), *va_copy*(), *va_end*(), and *va_start*() to the list of async-signal-safe functions and updating related text to apply to function-like macros.

Austin Group Defect 728 is applied, reducing the set of circumstances in which undefined behavior results when a signal handler refers to an object with static or thread storage duration.

Austin Group Defect 841 is applied, adding *pthread_setcancelstate*() to the list of async-signal-safe functions and making it implementation-defined which additional interfaces are also async-signal-safe.

Austin Group Defect 986 is applied, adding *strlcat*(), *strlcpy*(), *wcslcat*(), and *wcslcpy*() to the list of async-signal-safe functions.

Austin Group Defect 1138 is applied, adding the *sig2str*() function to the list of async-signal-safe functions.

Austin Group Defect 1141 is applied, changing ``core file'' to ``core image''.

Austin Group Defects 1142, 1455, and 1625 are applied, adding the *pread*(), *pwrite*(), *readv*(), *waitid*(), and *writev*() functions to the list of async-signal-safe functions.

Austin Group Defect 1151 is applied, adding the *tcgetwinsize*() and *tcsetwinsize*() functions to the list of async-signal-safe functions.

Austin Group Defect 1215 is applied, removing XSI shading from text relating to abnormal process termination with additional actions.

Austin Group Defect 1263 is applied, adding the *ppoll*() function to the list of async-signal-safe functions.

Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

Austin Group Defect 1667 is applied, adding *getresgid*(), *getresuid*(), *setresgid*(), and *setresuid*() to the list of async-signal-safe functions.

Austin Group Defect 1744 is applied, adding *killpg*() to the list of async-signal-safe functions.

B.2.4.4 *Signal Effects on Other Functions*

The most common behavior of an interrupted function after a signal-catching function returns is for the interrupted function to give an [EINTR] error unless the SA_RESTART flag is in effect for the signal. However, there are a number of specific exceptions, including *sleep*() and certain situations with *read*() and *write*().

The historical implementations of many functions defined by POSIX.1-2024 are not interruptible, but delay delivery of signals generated during their execution until after they complete. This is never a problem for functions that are guaranteed to complete in a short (imperceptible to a

human) period of time. It is normally those functions that can suspend a process indefinitely or for long periods of time (for example, *wait*( ), *pause*( ), *sigsuspend*( ), *sleep*( ), or *read*( )/*write*( ) on a slow device like a terminal) that are interruptible. This permits applications to respond to interactive signals or to set timeouts on calls to most such functions with *alarm*( ). Therefore, implementations should generally make such functions (including ones defined as extensions) interruptible.

Functions not mentioned explicitly as interruptible may be so on some implementations, possibly as an extension where the function gives an [EINTR] error. There are several functions (for example, *getpid*( ), *getuid*( )) that are specified as never returning an error, which can thus never be extended in this way.

If a signal-catching function returns while the SA_RESTART flag is in effect, an interrupted function is restarted at the point it was interrupted. Conforming applications cannot make assumptions about the internal behavior of interrupted functions, even if the functions are async-signal-safe. For example, suppose the *read*( ) function is interrupted with SA_RESTART in effect, the signal-catching function closes the file descriptor being read from and returns, and the *read*( ) function is then restarted; in this case the application cannot assume that the *read*( ) function will give an [EBADF] error, since *read*( ) might have checked the file descriptor for validity before being interrupted.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0017 [807] is applied.

B.2.5 Standard I/O Streams

Although the ISO C standard guarantees that, at program start-up, *stdin* is open for reading and *stdout* and *stderr* are open for writing, this guarantee is contingent (as are all guarantees made by the ISO C and POSIX standards) on the program being executed in a conforming environment. Programs executed with file descriptor 0 not open for reading or with file descriptor 1 or 2 not open for writing are executed in a non-conforming environment. Application writers are warned (in *exec*, *posix_spawn*( ), and Section C.2.7, on page 3890) not to execute a standard utility or a conforming application with file descriptor 0 not open for reading or with file descriptor 1 or 2 not open for writing.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0018 [608] is applied.

Austin Group Defect 689 is applied, clarifying the handling of deadlock situations when locking a stream.

Austin Group Defect 1144 is applied, clarifying the effect of *setvbuf*( ) on memory streams.

Austin Group Defect 1153 is applied, clarifying that the behavior is undefined if a memory buffer associated with a standard I/O stream overlaps with the destination buffer of a call that reads from the stream or with the source buffer of a call that writes to the stream.

Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

Austin Group Defect 1347 is applied, clarifying the requirements for how *stderr*, *stdin*, and *stdout* are opened at program start-up.

B.2.5.1 *Interaction of File Descriptors and Standard I/O Streams*

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0019 [480] is applied.

Austin Group Defect 1183 is applied, changing ``non-full'' to ``non-null''.

Austin Group Defect 1318 is applied, changing the list of functions that close file descriptors.

129299 *B.2.5.2 Stream Orientation and Encoding Rules*

129300 Austin Group Defect 1040 is applied, clarifying that conversion to or from (possibly multi-byte)
129301 characters is not performed by wide character I/O functions when the stream was opened using
129302 *open_wmemstream*( ).

129303 B.2.6 File Descriptor Allocation

129304 Functions such as *pipe*( ) and *socketpair*( ) which allocate two file descriptors are permitted to
129305 perform the two allocations independently. This means that other threads or signal handlers
129306 may perform operations on file descriptors in between the two allocations and this can result in
129307 the two file descriptors not having adjacent values or in the second allocation producing a lower
129308 value than the first.

129309 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0032 [835] is applied.

129310 B.2.7 XSI Interprocess Communication

129311 There are two forms of IPC supported as options in POSIX.1-2024. The traditional System V IPC
129312 routines derived from the SVID—that is, the *msg**( ), *sem**( ), and *shm**( ) interfaces—are
129313 mandatory on XSI-conformant systems. Thus, all XSI-conformant systems provide the same
129314 mechanisms for manipulating messages, shared memory, and semaphores.

129315 In addition, the POSIX Realtime Extension provides an alternate set of routines for those systems
129316 supporting the appropriate options.

129317 The application developer is presented with a choice: the System V interfaces or the POSIX
129318 interfaces (loosely derived from the Berkeley interfaces). The XSI profile prefers the System V
129319 interfaces, but the POSIX interfaces may be more suitable for realtime or other performance-
129320 sensitive applications.

129321 *B.2.7.1 IPC General Description*

129322 General information that is shared by all three mechanisms is described in this section. The
129323 common permissions mechanism is briefly introduced, describing the mode bits, and how they
129324 are used to determine whether or not a process has access to read or write/alter the appropriate
129325 instance of one of the IPC mechanisms. All other relevant information is contained in the
129326 reference pages themselves.

129327 The semaphore type of IPC allows processes to communicate through the exchange of
129328 semaphore values. A semaphore is a positive integer. Since many applications require the use of
129329 more than one semaphore, XSI-conformant systems have the ability to create sets or arrays of
129330 semaphores.

129331 Calls to support semaphores include:

129332 *semctl*( ), *semget*( ), *semop*( )

129333 Semaphore sets are created by using the *semget*( ) function.

129334 The message type of IPC allows processes to communicate through the exchange of data stored
129335 in buffers. This data is transmitted between processes in discrete portions known as messages.

129336 Calls to support message queues include:

129337 *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( )

The shared memory type of IPC allows two or more processes to share memory and consequently the data contained therein. This is done by allowing processes to set up access to a common memory address space. This sharing of memory provides a fast means of exchange of data between processes.

Calls to support shared memory include:

shmctl( ), *shmdt*( ), *shmget*( )

The *ftok*( ) interface is also provided.

Austin Group Defect 377 is applied, changing the table giving the values for the *mode* member of the **ipc_perm** structure.

B.2.8 Realtime

Advisory Information

POSIX.1b contains an Informative Annex with proposed interfaces for ``realtime files''. These interfaces could determine groups of the exact parameters required to do ``direct I/O'' or ``extents''. These interfaces were objected to by a significant portion of the balloting group as too complex. A conforming application had little chance of correctly navigating the large parameter space to match its desires to the system. In addition, they only applied to a new type of file (realtime files) and they told the implementation exactly what to do as opposed to advising the implementation on application behavior and letting it optimize for the system the (portable) application was running on. For example, it was not clear how a system that had a disk array should set its parameters.

There seemed to be several overall goals:

- Optimizing sequential access
- Optimizing caching behavior
- Optimizing I/O data transfer
- Preallocation

The advisory interfaces, *posix_fadvise*( ) and *posix_madvise*( ), satisfy the first two goals. The POSIX_FADV_SEQUENTIAL and POSIX_MADV_SEQUENTIAL advice tells the implementation to expect serial access. Typically the system will prefetch the next several serial accesses in order to overlap I/O. It may also free previously accessed serial data if memory is tight. If the application is not doing serial access it can use POSIX_FADV_WILLNEED and POSIX_MADV_WILLNEED to accomplish I/O overlap, as required. When the application advises POSIX_FADV_RANDOM or POSIX_MADV_RANDOM behavior, the implementation usually tries to fetch a minimum amount of data with each request and it does not expect much locality. POSIX_FADV_DONTNEED and POSIX_MADV_DONTNEED allow the system to free up caching resources as the data will not be required in the near future.

POSIX_FADV_NOREUSE tells the system that caching the specified data is not optimal. For file I/O, the transfer should go directly to the user buffer instead of being cached internally by the implementation. To portably perform direct disk I/O on all systems, the application must perform its I/O transfers according to the following rules:

1. The user buffer should be aligned according to the {POSIX_REC_XFER_ALIGN} *pathconf*( ) variable.

2. The number of bytes transferred in an I/O operation should be a multiple of the {POSIX_ALLOC_SIZE_MIN} *pathconf*( ) variable.
3. The offset into the file at the start of an I/O operation should be a multiple of the {POSIX_ALLOC_SIZE_MIN} *pathconf*( ) variable.
4. The application should ensure that all threads which open a given file specify POSIX_FADV_NOREUSE to be sure that there is no unexpected interaction between threads using buffered I/O and threads using direct I/O to the same file.

In some cases, a user buffer must be properly aligned in order to be transferred directly to/from the device. The {POSIX_REC_XFER_ALIGN} *pathconf*( ) variable tells the application the proper alignment.

The preallocation goal is met by the space control function, *posix_fallocate*( ). The application can use *posix_fallocate*( ) to guarantee no [ENOSPC] errors and to improve performance by prepaying any overhead required for block allocation.

Implementations may use information conveyed by a previous *posix_fadvise*( ) call to influence the manner in which allocation is performed. For example, if an application did the following calls:

```
fd = open("file");
posix_fadvise(fd, offset, len, POSIX_FADV_SEQUENTIAL);
posix_fallocate(fd, len, size);
```

an implementation might allocate the file contiguously on disk.

Finally, the *pathconf*( ) variables {POSIX_REC_MIN_XFER_SIZE}, {POSIX_REC_MAX_XFER_SIZE}, and {POSIX_REC_INCR_XFER_SIZE} tell the application a range of transfer sizes that are recommended for best I/O performance.

Where bounded response time is required, the vendor can supply the appropriate settings of the advisories to achieve a guaranteed performance level.

The interfaces meet the goals while allowing applications using regular files to take advantage of performance optimizations. The interfaces tell the implementation expected application behavior which the implementation can use to optimize performance on a particular system with a particular dynamic load.

The *posix_memalign*( ) function was added to allow for the allocation of specifically aligned buffers; for example, for {POSIX_REC_XFER_ALIGN}.

The working group also considered the alternative of adding a function which would return an aligned pointer to memory within a user-supplied buffer. This was not considered to be the best method, because it potentially wastes large amounts of memory when buffers need to be aligned on large alignment boundaries.

Message Passing

This section provides the rationale for the definition of the message passing interface in POSIX.1-2024. This is presented in terms of the objectives, models, and requirements imposed upon this interface.

- Objectives

  Many applications, including both realtime and database applications, require a means of passing arbitrary amounts of data between cooperating processes comprising the overall application on one or more processors. Many conventional interfaces for interprocess communication are insufficient for realtime applications in that efficient and deterministic

129423 data passing methods cannot be implemented. This has prompted the definition of
129424 message passing interfaces providing these facilities:

129425 — Open a message queue.

129426 — Send a message to a message queue.

129427 — Receive a message from a queue, either synchronously or asynchronously.

129428 — Alter message queue attributes for flow and resource control.

129429 It is assumed that an application may consist of multiple cooperating processes and that
129430 these processes may wish to communicate and coordinate their activities. The message
129431 passing facility described in POSIX.1-2024 allows processes to communicate through
129432 system-wide queues. These message queues are accessed through names that may be
129433 pathnames. A message queue can be opened for use by multiple sending and/or multiple
129434 receiving processes.

129435 • Background on Embedded Applications

129436 Interprocess communication utilizing message passing is a key facility for the construction
129437 of deterministic, high-performance realtime applications. The facility is present in all
129438 realtime systems and is the framework upon which the application is constructed. The
129439 performance of the facility is usually a direct indication of the performance of the resulting
129440 application.

129441 Realtime applications, especially for embedded systems, are typically designed around the
129442 performance constraints imposed by the message passing mechanisms. Applications for
129443 embedded systems are typically very tightly constrained. Application developers expect to
129444 design and control the entire system. In order to minimize system costs, the writer will
129445 attempt to use all resources to their utmost and minimize the requirement to add
129446 additional memory or processors.

129447 The embedded applications usually share address spaces and only a simple message
129448 passing mechanism is required. The application can readily access common data incurring
129449 only mutual-exclusion overheads. The models desired are the simplest possible with the
129450 application building higher-level facilities only when needed.

129451 • Requirements

129452 The following requirements determined the features of the message passing facilities
129453 defined in POSIX.1-2024:

129454 — Naming of Message Queues

129455 The mechanism for gaining access to a message queue is a pathname evaluated in a
129456 context that is allowed to be a file system name space, or it can be independent of
129457 any file system. This is a specific attempt to allow implementations based on either
129458 method in order to address both embedded systems and to also allow
129459 implementation in larger systems.

129460 The interface of *mq_open*() is defined to allow but not require the access control and
129461 name conflicts resulting from utilizing a file system for name resolution. All required
129462 behavior is specified for the access control case. Yet a conforming implementation,
129463 such as an embedded system kernel, may define that there are no distinctions
129464 between users and may define that all processes have all access privileges.

129465 — Embedded System Naming

129466 Embedded systems need to be able to utilize independent name spaces for accessing
129467 the various system objects. They typically do not have a file system, precluding its

utilization as a common name resolution mechanism. The modularity of an embedded system limits the connections between separate mechanisms that can be allowed.

Embedded systems typically do not have any access protection. Since the system does not support the mixing of applications from different areas, and usually does not even have the concept of an authorization entity, access control is not useful.

— Large System Naming

On systems with more functionality, the name resolution must support the ability to use the file system as the name resolution mechanism/object storage medium and to have control over access to the objects. Utilizing the pathname space can result in further errors when the names conflict with other objects.

— Fixed Size of Messages

The interfaces impose a fixed upper bound on the size of messages that can be sent to a specific message queue. The size is set on an individual queue basis and cannot be changed dynamically.

The purpose of the fixed size is to increase the ability of the system to optimize the implementation of *mq_send*() and *mq_receive*(). With fixed sizes of messages and fixed numbers of messages, specific message blocks can be pre-allocated. This eliminates a significant amount of checking for errors and boundary conditions. Additionally, an implementation can optimize data copying to maximize performance. Finally, with a restricted range of message sizes, an implementation is better able to provide deterministic operations.

— Prioritization of Messages

Message prioritization allows the application to determine the order in which messages are received. Prioritization of messages is a key facility that is provided by most realtime kernels and is heavily utilized by the applications. The major purpose of having priorities in message queues is to avoid priority inversions in the message system, where a high-priority message is delayed behind one or more lower-priority messages. This allows the applications to be designed so that they do not need to be interrupted in order to change the flow of control when exceptional conditions occur. The prioritization does add additional overhead to the message operations, in those cases it is actually used, but a clever implementation can optimize for the FIFO case to make that more efficient.

— Asynchronous Notification

The interface supports the ability to have a task asynchronously notified of the availability of a message on the queue. The purpose of this facility is to allow the task to perform other functions and yet still be notified that a message has become available on the queue.

To understand the requirement for this function, it is useful to understand two models of application design: a single task performing multiple functions and multiple tasks performing a single function. Each of these models has advantages.

Asynchronous notification is required to build the model of a single task performing multiple operations. This model typically results from either the expectation that interruption is less expensive than utilizing a separate task or from the growth of the application to include additional functions.

Semaphores

Semaphores are a high-performance process synchronization mechanism. Semaphores are named by null-terminated strings of characters.

A semaphore is created using the *sem_init*( ) function or the *sem_open*( ) function with the O_CREAT flag set in *oflag*.

To use a semaphore, a process has to first initialize the semaphore or inherit an open descriptor for the semaphore via *fork*( ).

A semaphore preserves its state when the last reference is closed. For example, if a semaphore has a value of 13 when the last reference is closed, it will have a value of 13 when it is next opened.

When a semaphore is created, an initial state for the semaphore has to be provided. This value is a non-negative integer. Negative values are not possible since they indicate the presence of blocked processes. The persistence of any of these objects across a system crash or a system reboot is undefined. Conforming applications must not depend on any sort of persistence across a system reboot or a system crash.

- Models and Requirements

  A realtime system requires synchronization and communication between the processes comprising the overall application. An efficient and reliable synchronization mechanism has to be provided in a realtime system that will allow more than one schedulable process mutually-exclusive access to the same resource. This synchronization mechanism has to allow for the optimal implementation of synchronization or systems implementors will define other, more cost-effective methods.

  At issue are the methods whereby multiple processes (tasks) can be designed and implemented to work together in order to perform a single function. This requires interprocess communication and synchronization. A semaphore mechanism is the lowest level of synchronization that can be provided by an operating system.

  A semaphore is defined as an object that has an integral value and a set of blocked processes associated with it. If the value is positive or zero, then the set of blocked processes is empty; otherwise, the size of the set is equal to the absolute value of the semaphore value. The value of the semaphore can be incremented or decremented by any process with access to the semaphore and must be done as an indivisible operation. When a semaphore value is less than or equal to zero, any process that attempts to lock it again will block or be informed that it is not possible to perform the operation.

  A semaphore may be used to guard access to any resource accessible by more than one schedulable task in the system. It is a global entity and not associated with any particular process. As such, a method of obtaining access to the semaphore has to be provided by the operating system. A process that wants access to a critical resource (section) has to wait on the semaphore that guards that resource. When the semaphore is locked on behalf of a process, it knows that it can utilize the resource without interference by any other cooperating process in the system. When the process finishes its operation on the resource, leaving it in a well-defined state, it posts the semaphore, indicating that some other process may now obtain the resource associated with that semaphore.

  In this section, mutexes and condition variables are specified as the synchronization mechanisms between threads.

  These primitives are typically used for synchronizing threads that share memory in a single process. However, this section provides an option allowing the use of these synchronization interfaces and objects between processes that share memory, regardless of

the method for sharing memory.

Much experience with semaphores shows that there are two distinct uses of synchronization: locking, which is typically of short duration; and waiting, which is typically of long or unbounded duration. These distinct usages map directly onto mutexes and condition variables, respectively.

Semaphores are provided in POSIX.1-2024 primarily to provide a means of synchronization for processes; these processes may or may not share memory. Mutexes and condition variables are specified as synchronization mechanisms between threads; these threads always share (some) memory. Both are synchronization paradigms that have been in widespread use for a number of years. Each set of primitives is particularly well matched to certain problems.

With respect to binary semaphores, experience has shown that condition variables and mutexes are easier to use for many synchronization problems than binary semaphores. The primary reason for this is the explicit appearance of a Boolean predicate that specifies when the condition wait is satisfied. This Boolean predicate terminates a loop, including the call to *pthread_cond_wait*(). As a result, extra wakeups are benign since the predicate governs whether the thread will actually proceed past the condition wait. With stateful primitives, such as binary semaphores, the wakeup in itself typically means that the wait is satisfied. The burden of ensuring correctness for such waits is thus placed on *all* signalers of the semaphore rather than on an *explicitly coded* Boolean predicate located at the condition wait. Experience has shown that the latter creates a major improvement in safety and ease-of-use.

Counting semaphores are well matched to dealing with producer/consumer problems, including those that might exist between threads of different processes, or between a signal handler and a thread. In the former case, there may be little or no memory shared by the processes; in the latter case, one is not communicating between co-equal threads, but between a thread and an interrupt-like entity. It is for these reasons that POSIX.1-2024 allows semaphores to be used by threads.

Mutexes and condition variables have been effectively used with and without priority inheritance, priority ceiling, and other attributes to synchronize threads that share memory. The efficiency of their implementation is comparable to or better than that of other synchronization primitives that are sometimes harder to use (for example, binary semaphores). Furthermore, there is at least one known implementation of Ada tasking that uses these primitives. Mutexes and condition variables together constitute an appropriate, sufficient, and complete set of inter-thread synchronization primitives.

Efficient multi-threaded applications require high-performance synchronization primitives. Considerations of efficiency and generality require a small set of primitives upon which more sophisticated synchronization functions can be built.

- Standardization Issues

It is possible to implement very high-performance semaphores using test-and-set instructions on shared memory locations. The library routines that implement such a high-performance interface have to properly ensure that a *sem_wait*() or *sem_trywait*() operation that cannot be performed will issue a blocking semaphore system call or properly report the condition to the application. The same interface to the application program would be provided by a high-performance implementation.

B.2.8.1 Realtime Signals

Realtime Signals Extension

This portion of the rationale presents models, requirements, and standardization issues relevant to the Realtime Signals Extension. This extension provides the capability required to support reliable, deterministic, asynchronous notification of events. While a new mechanism, unencumbered by the historical usage and semantics of POSIX.1 signals, might allow for a more efficient implementation, the application requirements for event notification can be met with a small number of extensions to signals. Therefore, a minimal set of extensions to signals to support the application requirements is specified.

The realtime signal extensions specified in this section are used by other realtime functions requiring asynchronous notification:

- Models

  The model supported is one of multiple cooperating processes, each of which handles multiple asynchronous external events. Events represent occurrences that are generated as the result of some activity in the system. Examples of occurrences that can constitute an event include:

  — Completion of an asynchronous I/O request

  — Expiration of a POSIX.1b timer

  — Arrival of an interprocess message

  — Generation of a user-defined event

  Processing of these events may occur synchronously via polling for event notifications or asynchronously via a software interrupt mechanism. Existing practice for this model is well established for traditional proprietary realtime operating systems, realtime executives, and realtime extended POSIX-like systems.

  A contrasting model is that of ``cooperating sequential processes'' where each process handles a single priority of events via polling. Each process blocks while waiting for events, and each process depends on the preemptive, priority-based process scheduling mechanism to arbitrate between events of different priority that need to be processed concurrently. Existing practice for this model is also well established for small realtime executives that typically execute in an unprotected physical address space, but it is just emerging in the context of a fuller function operating system with multiple virtual address spaces.

  It could be argued that the cooperating sequential process model, and the facilities supported by the POSIX Threads Extension obviate a software interrupt model. But, even with the cooperating sequential process model, the need has been recognized for a software interrupt model to handle exceptional conditions and process aborting, so the mechanism must be supported in any case. Furthermore, it is not the purview of POSIX.1-2024 to attempt to convince realtime practitioners that their current application models based on software interrupts are ``broken'' and should be replaced by the cooperating sequential process model. Rather, it is the charter of POSIX.1-2024 to provide standard extensions to mechanisms that support existing realtime practice.

- Requirements

  This section discusses the following realtime application requirements for asynchronous event notification:

— Reliable delivery of asynchronous event notification

The events notification mechanism guarantees delivery of an event notification. Asynchronous operations (such as asynchronous I/O and timers) that complete significantly after they are invoked have to guarantee that delivery of the event notification can occur at the time of completion.

— Prioritized handling of asynchronous event notifications

The events notification mechanism supports the assigning of a user function as an event notification handler. Furthermore, the mechanism supports the preemption of an event handler function by a higher priority event notification and supports the selection of the highest priority pending event notification when multiple notifications (of different priority) are pending simultaneously.

The model here is based on hardware interrupts. Asynchronous event handling allows the application to ensure that time-critical events are immediately processed when delivered, without the indeterminism of being at a random location within a polling loop. Use of handler priority allows the specification of how handlers are interrupted by other higher priority handlers.

— Differentiation between multiple occurrences of event notifications of the same type

The events notification mechanism passes an application-defined value to the event handler function. This value can be used for a variety of purposes, such as enabling the application to identify which of several possible events of the same type (for example, timer expirations) has occurred.

— Polled reception of asynchronous event notifications

The events notification mechanism supports blocking and non-blocking polls for asynchronous event notification.

The polled mode of operation is often preferred over the interrupt mode by those practitioners accustomed to this model. Providing support for this model facilitates the porting of applications based on this model to POSIX.1b conforming systems.

— Deterministic response to asynchronous event notifications

The events notification mechanism does not preclude implementations that provide deterministic event dispatch latency and minimizes the number of system calls needed to use the event facilities during realtime processing.

- Rationale for Extension

POSIX.1 signals have many of the characteristics necessary to support the asynchronous handling of event notifications, and the Realtime Signals Extension addresses the following deficiencies in the POSIX.1 signal mechanism:

— Signals do not support reliable delivery of event notification. Subsequent occurrences of a pending signal are not guaranteed to be delivered.

— Signals do not support prioritized delivery of event notifications. The order of signal delivery when multiple unblocked signals are pending is undefined.

— Signals do not support the differentiation between multiple signals of the same type.

B.2.8.2 Asynchronous I/O

Many applications need to interact with the I/O subsystem in an asynchronous manner. The asynchronous I/O mechanism provides the ability to overlap application processing and I/O operations initiated by the application. The asynchronous I/O mechanism allows a single process to perform I/O simultaneously to a single file multiple times or to multiple files multiple times.

Overview

Asynchronous I/O operations proceed in logical parallel with the processing done by the application after the asynchronous I/O has been initiated. Other than this difference, asynchronous I/O behaves similarly to normal I/O using *read*(), *write*(), *lseek*(), and *fsync*(). The effect of issuing an asynchronous I/O request is as if a separate thread of execution were to perform atomically the implied *lseek*() operation, if any, and then the requested I/O operation (either *read*(), *write*(), or *fsync*()). There is no seek implied with a call to *aio_fsync*(). Concurrent asynchronous operations and synchronous operations applied to the same file update the file as if the I/O operations had proceeded serially.

When asynchronous I/O completes, a signal can be delivered to the application to indicate the completion of the I/O. This signal can be used to indicate that buffers and control blocks used for asynchronous I/O can be reused. Signal delivery is not required for an asynchronous operation and may be turned off on a per-operation basis by the application. Signals may also be synchronously polled using *aio_suspend*(), *sigtimedwait*(), or *sigwaitinfo*().

Normal I/O has a return value and an error status associated with it. Asynchronous I/O returns a value and an error status when the operation is first submitted, but that only relates to whether the operation was successfully queued up for servicing. The I/O operation itself also has a return status and an error value. To allow the application to retrieve the return status and the error value, functions are provided that, given the address of an asynchronous I/O control block, yield the return and error status associated with the operation. Until an asynchronous I/O operation is done, its error status is [EINPROGRESS]. Thus, an application can poll for completion of an asynchronous I/O operation by waiting for the error status to become equal to a value other than [EINPROGRESS]. The return status of an asynchronous I/O operation is undefined so long as the error status is equal to [EINPROGRESS].

Storage for asynchronous operation return and error status may be limited. Submission of asynchronous I/O operations may fail if this storage is exceeded. When an application retrieves the return status of a given asynchronous operation, therefore, any system-maintained storage used for this status and the error status may be reclaimed for use by other asynchronous operations.

Asynchronous I/O can be performed on file descriptors that have been enabled for POSIX.1b synchronized I/O. In this case, the I/O operation still occurs asynchronously, as defined herein; however, the asynchronous operation I/O in this case is not completed until the I/O has reached either the state of synchronized I/O data integrity completion or synchronized I/O file integrity completion, depending on the sort of synchronized I/O that is enabled on the file descriptor.

Models

Three models illustrate the use of asynchronous I/O: a journalization model, a data acquisition model, and a model of the use of asynchronous I/O in supercomputing applications.

- Journalization Model

  Many realtime applications perform low-priority journalizing functions. Journalizing requires that logging records be queued for output without blocking the initiating process.

- Data Acquisition Model

  A data acquisition process may also serve as a model. The process has two or more channels delivering intermittent data that must be read within a certain time. The process issues one asynchronous read on each channel. When one of the channels needs data collection, the process reads the data and posts it through an asynchronous write to secondary memory for future processing.

- Supercomputing Model

  The supercomputing community has used asynchronous I/O much like that specified in POSIX.1 for many years. This community requires the ability to perform multiple I/O operations to multiple devices with a minimal number of entries to ``the system''; each entry to ``the system'' provokes a major delay in operations when compared to the normal progress made by the application. This existing practice motivated the use of combined *lseek*( ) and *read*( ) or *write*( ) calls, as well as the *lio_listio*( ) call. Another common practice is to disable signal notification for I/O completion, and simply poll for I/O completion at some interval by which the I/O should be completed. Likewise, interfaces like *aio_cancel*( ) have been in successful commercial use for many years. Note also that an underlying implementation of asynchronous I/O will require the ability, at least internally, to cancel outstanding asynchronous I/O, at least when the process exits. (Consider an asynchronous read from a terminal, when the process intends to exit immediately.)

Requirements

Asynchronous input and output for realtime implementations have these requirements:

- The ability to queue multiple asynchronous read and write operations to a single open instance. Both sequential and random access should be supported.
- The ability to queue asynchronous read and write operations to multiple open instances.
- The ability to obtain completion status information by polling and/or asynchronous event notification.
- Asynchronous event notification on asynchronous I/O completion is optional.
- It has to be possible for the application to associate the event with the *aiocbp* for the operation that generated the event.
- The ability to cancel queued requests.
- The ability to wait upon asynchronous I/O completion in conjunction with other types of events.
- The ability to accept an *aio_read*( ) and an *aio_cancel*( ) for a device that accepts a *read*( ), and the ability to accept an *aio_write*( ) and an *aio_cancel*( ) for a device that accepts a *write*( ). This does not imply that the operation is asynchronous.

Standardization Issues

The following issues are addressed by the standardization of asynchronous I/O:

- Rationale for New Interface

  Non-blocking I/O does not satisfy the needs of either realtime or high-performance computing models; these models require that a process overlap program execution and I/O processing. Realtime applications will often make use of direct I/O to or from the address space of the process, or require synchronized (unbuffered) I/O; they also require the ability to overlap this I/O with other computation. In addition, asynchronous I/O allows an application to keep a device busy at all times, possibly achieving greater throughput. Supercomputing and database architectures will often have specialized hardware that can provide true asynchrony underlying the logical asynchrony provided by this interface. In addition, asynchronous I/O should be supported by all types of files and devices in the same manner.

- Effect of Buffering

  If asynchronous I/O is performed on a file that is buffered prior to being actually written to the device, it is possible that asynchronous I/O will offer no performance advantage over normal I/O; the cycles *stolen* to perform the asynchronous I/O will be taken away from the running process and the I/O will occur at interrupt time. This potential lack of gain in performance in no way obviates the need for asynchronous I/O by realtime applications, which very often will use specialized hardware support, multiple processors, and/or unbuffered, synchronized I/O.

B.2.8.3 Memory Management

All memory management and shared memory definitions are located in the **<sys/mman.h>** header. This is for alignment with historical practice.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/7 is applied, correcting the shading and margin markers in the introduction to Section 2.8.3.1.

Memory Locking Functions

This portion of the rationale presents models, requirements, and standardization issues relevant to process memory locking.

- Models

  Realtime systems that conform to POSIX.1-2024 are expected (and desired) to be supported on systems with demand-paged virtual memory management, non-paged swapping memory management, and physical memory systems with no memory management hardware. The general case, however, is the demand-paged, virtual memory system with each POSIX process running in a virtual address space. Note that this includes architectures where each process resides in its own virtual address space and architectures where the address space of each process is only a portion of a larger global virtual address space.

  The concept of memory locking is introduced to eliminate the indeterminacy introduced by paging and swapping, and to support an upper bound on the time required to access the memory mapped into the address space of a process. Ideally, this upper bound will be the same as the time required for the processor to access ``main memory'', including any address translation and cache miss overheads. But some implementations—primarily on mainframes—will not actually force locked pages to be loaded and held resident in main memory. Rather, they will handle locked pages so that accesses to these pages will meet the

129815 performance metrics for locked process memory in the implementation. Also, although it
129816 is not, for example, the intention that this interface, as specified, be used to lock process
129817 memory into ``cache'', it is conceivable that an implementation could support a large static
129818 RAM memory and define this as ``main memory'' and use a large[r] dynamic RAM as
129819 ``backing store''. These interfaces could then be interpreted as supporting the locking of
129820 process memory into the static RAM. Support for multiple levels of backing store would
129821 require extensions to these interfaces.

129822 Implementations may also use memory locking to guarantee a fixed translation between
129823 virtual and physical addresses where such is beneficial to improving determinacy for
129824 direct-to/from-process input/output. POSIX.1-2024 does not guarantee to the application
129825 that the virtual-to-physical address translations, if such exist, are fixed, because such
129826 behavior would not be implementable on all architectures on which implementations of
129827 POSIX.1-2024 are expected. But POSIX.1-2024 does mandate that an implementation
129828 define, for the benefit of potential users, whether or not locking guarantees fixed
129829 translations.

129830 Memory locking is defined with respect to the address space of a process. Only the pages
129831 mapped into the address space of a process may be locked by the process, and when the
129832 pages are no longer mapped into the address space—for whatever reason—the locks
129833 established with respect to that address space are removed. Shared memory areas warrant
129834 special mention, as they may be mapped into more than one address space or mapped
129835 more than once into the address space of a process; locks may be established on pages
129836 within these areas with respect to several of these mappings. In such a case, the lock state
129837 of the underlying physical pages is the logical OR of the lock state with respect to each of
129838 the mappings. Only when all such locks have been removed are the shared pages
129839 considered unlocked.

129840 In recognition of the page granularity of Memory Management Units (MMU), and in order
129841 to support locking of ranges of address space, memory locking is defined in terms of
129842 ``page'' granularity. That is, for the interfaces that support an address and size specification
129843 for the region to be locked, the address must be on a page boundary, and all pages mapped
129844 by the specified range are locked, if valid. This means that the length is implicitly rounded
129845 up to a multiple of the page size. The page size is implementation-defined and is available
129846 to applications as a compile-time symbolic constant or at runtime via *sysconf*( ).

129847 A ``real memory'' POSIX.1b implementation that has no MMU could elect not to support
129848 these interfaces, returning [ENOSYS]. But an application could easily interpret this as
129849 meaning that the implementation would unconditionally page or swap the application
129850 when such is not the case. It is the intention of POSIX.1-2024 that such a system could
129851 define these interfaces as ``NO-OPs'', returning success without actually performing any
129852 function except for mandated argument checking.

129853 • Requirements

129854 For realtime applications, memory locking is generally considered to be required as part of
129855 application initialization. This locking is performed after an application has been loaded
129856 (that is, *exec*'d) and the program remains locked for its entire lifetime. But to support
129857 applications that undergo major mode changes where, in one mode, locking is required,
129858 but in another it is not, the specified interfaces allow repeated locking and unlocking of
129859 memory within the lifetime of a process.

129860 When a realtime application locks its address space, it should not be necessary for the
129861 application to then ``touch'' all of the pages in the address space to guarantee that they are
129862 resident or else suffer potential paging delays the first time the page is referenced. Thus,
129863 POSIX.1-2024 requires that the pages locked by the specified interfaces be resident when

the locking functions return successfully.

Many architectures support system-managed stacks that grow automatically when the current extent of the stack is exceeded. A realtime application has a requirement to be able to ``preallocate'' sufficient stack space and lock it down so that it will not suffer page faults to grow the stack during critical realtime operation. There was no consensus on a portable way to specify how much stack space is needed, so POSIX.1-2024 supports no specific interface for preallocating stack space. But an application can portably lock down a specific amount of stack space by specifying MCL_FUTURE in a call to *mlockall*( ) and then calling a dummy function that declares an automatic array of the desired size.

Memory locking for realtime applications is also generally considered to be an ``all or nothing'' proposition. That is, the entire process, or none, is locked down. But, for applications that have well-defined sections that need to be locked and others that do not, POSIX.1-2024 supports an optional set of interfaces to lock or unlock a range of process addresses. Reasons for locking down a specific range include:

— An asynchronous event handler function that must respond to external events in a deterministic manner such that page faults cannot be tolerated

— An input/output ``buffer'' area that is the target for direct-to-process I/O, and the overhead of implicit locking and unlocking for each I/O call cannot be tolerated

Finally, locking is generally viewed as an ``application-wide'' function. That is, the application is globally aware of which regions are locked and which are not over time. This is in contrast to a function that is used temporarily within a ``third party'' library routine whose function is unknown to the application, and therefore must have no ``side-effects''. The specified interfaces, therefore, do not support ``lock stacking'' or ``lock nesting'' within a process. But, for pages that are shared between processes or mapped more than once into a process address space, ``lock stacking'' is essentially mandated by the requirement that unlocking of pages that are mapped by more that one process or more than once by the same process does not affect locks established on the other mappings.

There was some support for ``lock stacking'' so that locking could be transparently used in functions or opaque modules. But the consensus was not to burden all implementations with lock stacking (and reference counting), and an implementation option was proposed. There were strong objections to the option because applications would have to support both options in order to remain portable. The consensus was to eliminate lock stacking altogether, primarily through overwhelming support for the System V ``m[un]lock[all]'' interface on which POSIX.1-2024 is now based.

Locks are not inherited across *fork*( )s because some implementations implement *fork*( ) by creating new address spaces for the child. In such an implementation, requiring locks to be inherited would lead to new situations in which a fork would fail due to the inability of the system to lock sufficient memory to lock both the parent and the child. The consensus was that there was no benefit to such inheritance. Note that this does not mean that locks are removed when, for instance, a thread is created in the same address space.

Similarly, locks are not inherited across *exec* because some implementations implement *exec* by unmapping all of the pages in the address space (which, by definition, removes the locks on these pages), and maps in pages of the *exec*'d image. In such an implementation, requiring locks to be inherited would lead to new situations in which *exec* would fail. Reporting this failure would be very cumbersome to detect in time to report to the calling process, and no appropriate mechanism exists for informing the *exec*'d process of its status.

It was determined that, if the newly loaded application required locking, it was the responsibility of that application to establish the locks. This is also in keeping with the

general view that it is the responsibility of the application to be aware of all locks that are established.

There was one request to allow (not mandate) locks to be inherited across *fork*(), and a request for a flag, MCL_INHERIT, that would specify inheritance of memory locks across *exec*s. Given the difficulties raised by this and the general lack of support for the feature in POSIX.1-2024, it was not added. POSIX.1-2024 does not preclude an implementation from providing this feature for administrative purposes, such as a ``run'' command that will lock down and execute a specified application. Additionally, the rationale for the objection equated *fork*() with creating a thread in the address space. POSIX.1-2024 does not mandate releasing locks when creating additional threads in an existing process.

- Standardization Issues

One goal of POSIX.1-2024 is to define a set of primitives that provide the necessary functionality for realtime applications, with consideration for the needs of other application domains where such were identified, which is based to the extent possible on existing industry practice.

The Memory Locking option is required by many realtime applications to tune performance. Such a facility is accomplished by placing constraints on the virtual memory system to limit paging of time of the process or of critical sections of the process. This facility should not be used by most non-realtime applications.

Optional features provided in POSIX.1-2024 allow applications to lock selected address ranges with the caveat that the process is responsible for being aware of the page granularity of locking and the unnested nature of the locks.

Mapped Files Functions

The memory mapped files functionality provides a mechanism that allows a process to access files by directly incorporating file data into its address space. Once a file is ``mapped'' into a process address space, the data can be manipulated by instructions as memory. The use of mapped files can significantly reduce I/O data movement since file data does not have to be copied into process data buffers as in *read*() and *write*(). If more than one process maps a file, its contents are shared among them. This provides a low overhead mechanism by which processes can synchronize and communicate.

- Historical Perspective

Realtime applications have historically been implemented using a collection of cooperating processes or tasks. In early systems, these processes ran on bare hardware (that is, without an operating system) with no memory relocation or protection. The application paradigms that arose from this environment involve the sharing of data between the processes.

When realtime systems were implemented on top of vendor-supplied operating systems, the paradigm or performance benefits of direct access to data by multiple processes was still deemed necessary. As a result, operating systems that claim to support realtime applications must support the shared memory paradigm.

Additionally, a number of realtime systems provide the ability to map specific sections of the physical address space into the address space of a process. This ability is required if an application is to obtain direct access to memory locations that have specific properties (for example, refresh buffers or display devices, dual ported memory locations, DMA target locations). The use of this ability is common enough to warrant some degree of standardization of its interface. This ability overlaps the general paradigm of shared memory in that, in both instances, common global objects are made addressable by individual processes or tasks.

Finally, a number of systems also provide the ability to map process addresses to files. This provides both a general means of sharing persistent objects, and using files in a manner that optimizes memory and swapping space usage.

Simple shared memory is clearly a special case of the more general file mapping capability. In addition, there is relatively widespread agreement and implementation of the file mapping interface. In these systems, many different types of objects can be mapped (for example, files, memory, devices, and so on) using the same mapping interfaces. This approach both minimizes interface proliferation and maximizes the generality of programs using the mapping interfaces.

- Memory Mapped Files Usage

A memory object can be concurrently mapped into the address space of one or more processes. The *mmap*() and *munmap*() functions allow a process to manipulate their address space by mapping portions of memory objects into it and removing them from it. When multiple processes map the same memory object, they can share access to the underlying data. Implementations may restrict the size and alignment of mappings to be on *page*-size boundaries. The page size, in bytes, is the value of the system-configurable variable {PAGESIZE}, typically accessed by calling *sysconf*() with a *name* argument of _SC_PAGESIZE. If an implementation has no restrictions on size or alignment, it may specify a 1-byte page size.

To map memory, a process first opens a memory object. The *ftruncate*() function can be used to contract or extend the size of the memory object even when the object is currently mapped. If the memory object is extended, the contents of the extended areas are zeros.

After opening a memory object, the application maps the object into its address space using the *mmap*() function call. Once a mapping has been established, it remains mapped until unmapped with *munmap*(), even if the memory object is closed. The *mprotect*() function can be used to change the memory protections initially established by *mmap*().

A *close*() of the file descriptor, while invalidating the file descriptor itself, does not unmap any mappings established for the memory object. The address space, including all mapped regions, is inherited on *fork*(). The entire address space is unmapped on process termination or by successful calls to any of the *exec* family of functions.

The *msync*() function is used to force mapped file data to permanent storage.

- Effects on Other Functions

With memory mapped files, the operation of the *open*(), *creat*(), and *unlink*() functions are a natural result of using the file system name space to map the global names for memory objects.

The *ftruncate*() function can be used to set the length of a sharable memory object.

The meaning of *stat*() fields other than the size and protection information is undefined on implementations where memory objects are not implemented using regular files. When regular files are used, the times reflect when the implementation updated the file image of the data, not when a process updated the data in memory.

The operations of *fdopen*(), *write*(), *read*(), and *lseek*() were made unspecified for objects opened with *shm_open*(), so that implementations that did not implement memory objects as regular files would not have to support the operation of these functions on shared memory objects.

The behavior of memory objects with respect to *close*(), *dup*(), *dup2*(), *dup3*(), *open*(), *close*(), *fork*(), *_exit*(), and the *exec* family of functions is the same as the behavior of the

existing practice of the *mmap*( ) function.

A memory object can still be referenced after a close. That is, any mappings made to the file are still in effect, and reads and writes that are made to those mappings are still valid and are shared with other processes that have the same mapping. Likewise, the memory object can still be used if any references remain after its name(s) have been deleted. Any references that remain after a close must not appear to the application as file descriptors.

This is existing practice for *mmap*( ) and *close*( ). In addition, there are already mappings present (text, data, stack) that do not have open file descriptors. The text mapping in particular is considered a reference to the file containing the text. The desire was to treat all mappings by the process uniformly. Also, many modern implementations use *mmap*( ) to implement shared libraries, and it would not be desirable to keep file descriptors for each of the many libraries an application can use. It was felt there were many other existing programs that used this behavior to free a file descriptor, and thus POSIX.1-2024 could not forbid it and still claim to be using existing practice.

For implementations that implement memory objects using memory only, memory objects will retain the memory allocated to the file after the last close and will use that same memory on the next open. Note that closing the memory object is not the same as deleting the name, since the memory object is still defined in the memory object name space.

The locks of *fcntl*( ) do not block any read or write operation, including read or write access to shared memory or mapped files. In addition, implementations that only support shared memory objects should not be required to implement record locks. The reference to *fcntl*( ) is added to make this point explicitly. The other *fcntl*( ) commands are useful with shared memory objects.

The size of pages that mapping hardware may be able to support may be a configurable value, or it may change based on hardware implementations. The addition of the _SC_PAGESIZE parameter to the *sysconf*( ) function is provided for determining the mapping page size at runtime.

Shared Memory Functions

Implementations may support the Shared Memory Objects option independently of memory mapped files. Shared memory objects are named regions of storage that may be independent of the file system and can be mapped into the address space of one or more processes to allow them to share the associated memory.

- Requirements

  Shared memory is used to share data among several processes, each potentially running at different priority levels, responding to different inputs, or performing separate tasks. Shared memory is not just simply providing common access to data, it is providing the fastest possible communication between the processes. With one memory write operation, a process can pass information to as many processes as have the memory region mapped.

  As a result, shared memory provides a mechanism that can be used for all other interprocess communication facilities. It may also be used by an application for implementing more sophisticated mechanisms than semaphores and message queues.

  The need for a shared memory interface is obvious for virtual memory systems, where the operating system is directly preventing processes from accessing each other's data. However, in unprotected systems, such as those found in some embedded controllers, a shared memory interface is needed to provide a portable mechanism to allocate a region of memory to be shared and then to communicate the address of that region to other processes.

130052 This, then, provides the minimum functionality that a shared memory interface must have
130053 in order to support realtime applications: to allocate and name an object to be mapped into
130054 memory for potential sharing (*open*( ) or *shm_open*( )), and to make the memory object
130055 available within the address space of a process (*mmap*( )). To complete the interface, a
130056 mechanism to release the claim of a process on a shared memory object (*munmap*( )) is also
130057 needed, as well as a mechanism for deleting the name of a sharable object that was
130058 previously created (*unlink*( ) or *shm_unlink*( )).

130059 After a mapping has been established, an implementation should not have to provide
130060 services to maintain that mapping. All memory writes into that area will appear
130061 immediately in the memory mapping of that region by any other processes.

130062 Thus, requirements include:

130063 — Support creation of sharable memory objects and the mapping of these objects into
130064 the address space of a process.

130065 — Sharable memory objects should be accessed by global names accessible from all
130066 processes.

130067 — Support the mapping of specific sections of physical address space (such as a
130068 memory mapped device) into the address space of a process. This should not be
130069 done by the process specifying the actual address, but again by an implementation-
130070 defined global name (such as a special device name) dedicated to this purpose.

130071 — Support the mapping of discrete portions of these memory objects.

130072 — Support for minimum hardware configurations that contain no physical media on
130073 which to store shared memory contents permanently.

130074 — The ability to preallocate the entire shared memory region so that minimum
130075 hardware configurations without virtual memory support can guarantee contiguous
130076 space.

130077 — The maximizing of performance by not requiring functionality that would require
130078 implementation interaction above creating the shared memory area and returning
130079 the mapping.

130080 Note that the above requirements do not preclude:

130081 — The sharable memory object from being implemented using actual files on an actual
130082 file system.

130083 — The global name that is accessible from all processes being restricted to a file system
130084 area that is dedicated to handling shared memory.

130085 — An implementation not providing implementation-defined global names for the
130086 purpose of physical address mapping.

130087 • Shared Memory Objects Usage

130088 If the Shared Memory Objects option is supported, a shared memory object may be
130089 created, or opened if it already exists, with the *shm_open*( ) function. If the shared memory
130090 object is created, it has a length of zero. The *ftruncate*( ) function can be used to set the size
130091 of the shared memory object after creation. The *shm_unlink*( ) function removes the name
130092 for a shared memory object created by *shm_open*( ).

130093 • Shared Memory Overview

130094 The shared memory facility defined by POSIX.1-2024 usually results in memory locations
130095 being added to the address space of the process. The implementation returns the address

of the new space to the application by means of a pointer. This works well in languages like C. However, in languages without pointer types it will not work. In the bindings for such a language, either a special COMMON section will need to be defined (which is unlikely), or the binding will have to allow existing structures to be mapped. The implementation will likely have to place restrictions on the size and alignment of such structures or will have to map a suitable region of the address space of the process into the memory object, and thus into other processes. These are issues for that particular language binding. For POSIX.1-2024, however, the practice will not be forbidden, merely undefined.

Two potentially different name spaces are used for naming objects that may be mapped into process address spaces. When using memory mapped files, files may be accessed via *open*( ). When the Shared Memory Objects option is supported, sharable memory objects that might not be files may be accessed via the *shm_open*( ) function. These operations are not mutually-exclusive.

Some implementations supporting the Shared Memory Objects option may choose to implement the shared memory object name space as part of the file system name space. There are several reasons for this:

— It allows applications to prevent name conflicts by use of the directory structure.

— It uses an existing mechanism for accessing global objects and prevents the creation of a new mechanism for naming global objects.

In such implementations, memory objects can be implemented using regular files, if that is what the implementation chooses. The *shm_open*( ) function can be implemented as an *open*( ) call in a fixed directory with the O_CLOEXEC flag set. The *shm_unlink*( ) function can be implemented as an *unlink*( ) call.

On the other hand, it is also expected that small embedded systems that support the Shared Memory Objects option may wish to implement shared memory without having any file systems present. In this case, the implementations may choose to use a simple string valued name space for shared memory regions. The *shm_open*( ) function permits either type of implementation.

Some implementations have hardware that supports protection of mapped data from certain classes of access and some do not. Systems that supply this functionality support the memory protection functionality.

Some implementations restrict size, alignment, and protections to be on *page*-size boundaries. If an implementation has no restrictions on size or alignment, it may specify a 1-byte page size. Applications on implementations that do support larger pages must be cognizant of the page size since this is the alignment and protection boundary.

Simple embedded implementations may have a 1-byte page size and only support the Shared Memory Objects option. This provides simple shared memory between processes without requiring mapping hardware.

POSIX.1-2024 specifically allows a memory object to remain referenced after a close because that is existing practice for the *mmap*( ) function.

130136 **Typed Memory Functions**

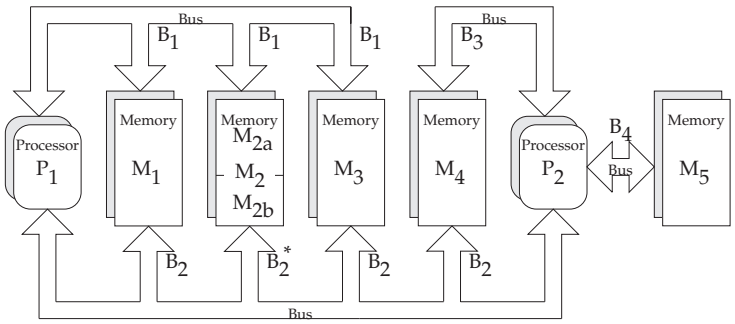
130137 Implementations may support the Typed Memory Objects option without supporting either the
130138 Shared Memory option or memory mapped files. Types memory objects are pools of specialized
130139 storage, different from the main memory resource normally used by a processor to hold code
130140 and data, that can be mapped into the address space of one or more processes.

130141 • Model

130142 Realtime systems conforming to one of the POSIX.13 realtime profiles are expected (and
130143 desired) to be supported on systems with more than one type or pool of memory (for
130144 example, SRAM, DRAM, ROM, EPROM, EEPROM), where each type or pool of memory
130145 may be accessible by one or more processors via one or more buses (ports). Memory
130146 mapped files, shared memory objects, and the language-specific storage allocation
130147 operators (*malloc*( ) for the ISO C standard, *new* for ISO Ada) fail to provide application
130148 program interfaces versatile enough to allow applications to control their utilization of
130149 such diverse memory resources. The typed memory interfaces *posix_typed_mem_open*( ),
130150 *posix_mem_offset*( ), *posix_typed_mem_get_info*( ), *mmap*( ), and *munmap*( ) defined herein
130151 support the model of typed memory described below.

130152 For purposes of this model, a system comprises several processors (for example, P_1 and
130153 P_2), several physical memory pools (for example, M_1, M_2, M_{2a}, M_{2b}, M_3, M_4, and M_5), and
130154 several buses or ``ports'' (for example, B_1, B_2, B_3, and B_4) interconnecting the various
130155 processors and memory pools in some system-specific way. Notice that some memory
130156 pools may be contained in others (for example, M_{2a} and M_{2b} are contained in M_2).

130157 Figure B-1 shows an example of such a model. In a system like this, an application should
130158 be able to perform the following operations:



* All addresses in pool M_2 (comprising pools M_{2a} and M_{2b}) accessible via port B_1.
Addresses in pool M_{2b} are also accessible via port B_2.
Addresses in pool M_{2a} are *not* accessible via port B_2.

130159 **Figure B-1** Example of a System with Typed Memory

130160 — Typed Memory Allocation

130161 An application should be able to allocate memory dynamically from the desired pool
130162 using the desired bus, and map it into the address space of a process. For example,
130163 processor P_1 can allocate some portion of memory pool M_1 through port B_1, treating
130164 all unmapped subareas of M_1 as a heap-storage resource from which memory may be
130165 allocated. This portion of memory is mapped into address space of the process, and
130166 subsequently deallocated when unmapped from all processes.

— Using the Same Storage Region from Different Buses

An application process with a mapped region of storage that is accessed from one bus should be able to map that same storage area at another address (subject to page size restrictions detailed in *mmap*()), to allow it to be accessed from another bus. For example, processor P_1 may wish to access the same region of memory pool M_{2b} both through ports B_1 and B_2.

— Sharing Typed Memory Regions

Several application processes running on the same or different processors may wish to share a particular region of a typed memory pool. Each process or processor may wish to access this region through different buses. For example, processor P_1 may want to share a region of memory pool M_4 with processor P_2, and they may be required to use buses B_2 and B_3, respectively, to minimize bus contention. A problem arises here when a process allocates and maps a portion of fragmented memory and then wants to share this region of memory with another process, either in the same processor or different processors. The solution adopted is to allow the first process to find out the memory map (offsets and lengths) of all the different fragments of memory that were mapped into its address space, by repeatedly calling *posix_mem_offset*(). Then, this process can pass the offsets and lengths obtained to the second process, which can then map the same memory fragments into its address space.

— Contiguous Allocation

The problem of finding the memory map of the different fragments of the memory pool that were mapped into logically contiguous addresses of a given process can be solved by requesting contiguous allocation. For example, a process in P_1 can allocate 10 Kbytes of physically contiguous memory from M_3-B_1, and obtain the offset (within pool M_3) of this block of memory. Then, it can pass this offset (and the length) to a process in P_2 using some interprocess communication mechanism. The second process can map the same block of memory by using the offset transferred and specifying M_3-B_2.

— Unallocated Mapping

Any subarea of a memory pool that is mapped to a process, either as the result of an allocation request or an explicit mapping, is normally unavailable for allocation. Special processes such as debuggers, however, may need to map large areas of a typed memory pool, yet leave those areas available for allocation.

Typed memory allocation and mapping has to coexist with storage allocation operators like *malloc*(), but systems are free to choose how to implement this coexistence. For example, it may be system configuration-dependent if all available system memory is made part of one of the typed memory pools or if some part will be restricted to conventional allocation operators. Equally system configuration-dependent may be the availability of operators like *malloc*() to allocate storage from certain typed memory pools. It is not excluded to configure a system such that a given named pool, P_1, is in turn split into non-overlapping named subpools. For example, M_1-B_1, M_2-B_1, and M_3-B_1 could also be accessed as one common pool M_{123}-B_1. A call to *malloc*() on P_1 could work on such a larger pool while full optimization of memory usage by P_1 would require typed memory allocation at the subpool level.

- Existing Practice

OS-9 provides for the naming (numbering) and prioritization of memory types by a system administrator. It then provides APIs to request memory allocation of typed (colored)

memory by number, and to generate a bus address from a mapped memory address (translate). When requesting colored memory, the user can specify type 0 to signify allocation from the first available type in priority order.

HP-RT presents interfaces to map different kinds of storage regions that are visible through a VME bus, although it does not provide allocation operations. It also provides functions to perform address translation between VME addresses and virtual addresses. It represents a VME-bus unique solution to the general problem.

The PSOS approach is similar (that is, based on a pre-established mapping of bus address ranges to specific memories) with a concept of segments and regions (regions dynamically allocated from a heap which is a special segment). Therefore, PSOS does not fully address the general allocation problem either. PSOS does not have a ``process''-based model, but more of a ``thread''-only-based model of multi-tasking. So mapping to a process address space is not an issue.

QNX uses the System V approach of opening specially named devices (shared memory segments) and using *mmap*() to then gain access from the process. They do not address allocation directly, but once typed shared memory can be mapped, an ``allocation manager'' process could be written to handle requests for allocation.

The System V approach also included allocation, implemented by opening yet other special ``devices'' which allocate, rather than appearing as a whole memory object.

The Orkid realtime kernel interface definition has operations to manage memory ``regions'' and ``pools'', which are areas of memory that may reflect the differing physical nature of the memory. Operations to allocate memory from these regions and pools are also provided.

- Requirements

Existing practice in SVID-derived UNIX systems relies on functionality similar to *mmap*() and its related interfaces to achieve mapping and allocation of typed memory. However, the issue of sharing typed memory (allocated or mapped) and the complication of multiple ports are not addressed in any consistent way by existing UNIX system practice. Part of this functionality is existing practice in specialized realtime operating systems. In order to solidify the capabilities implied by the model above, the following requirements are imposed on the interface:

— Identification of Typed Memory Pools and Ports

All processes (running in all processors) in the system are able to identify a particular (system configured) typed memory pool accessed through a particular (system configured) port by a name. That name is a member of a name space common to all these processes, but need not be the same name space as that containing ordinary pathnames. The association between memory pools/ports and corresponding names is typically established when the system is configured. The ``open'' operation for typed memory objects should be distinct from the *open*() function, for consistency with other similar services, but implementable on top of *open*(). This implies that the handle for a typed memory object will be a file descriptor.

— Allocation and Mapping of Typed Memory

Once a typed memory object has been identified by a process, it is possible to both map user-selected subareas of that object into process address space and to map system-selected (that is, dynamically allocated) subareas of that object, with user-specified length, into process address space. It is also possible to determine the maximum length of memory allocation that may be requested from a given typed

130262 memory object.

130263 — Sharing Typed Memory

130264 Two or more processes are able to share portions of typed memory, either user-
130265 selected or dynamically allocated. This requirement applies also to dynamically
130266 allocated regions of memory that are composed of several non-contiguous pieces.

130267 — Contiguous Allocation

130268 For dynamic allocation, it is the user's option whether the system is required to
130269 allocate a contiguous subarea within the typed memory object, or whether it is
130270 permitted to allocate discontiguous fragments which appear contiguous in the
130271 process mapping. Contiguous allocation simplifies the process of sharing allocated
130272 typed memory, while discontiguous allocation allows for potentially better recovery
130273 of deallocated typed memory.

130274 — Accessing Typed Memory Through Different Ports

130275 Once a subarea of a typed memory object has been mapped, it is possible to
130276 determine the location and length corresponding to a user-selected portion of that
130277 object within the memory pool. This location and length can then be used to remap
130278 that portion of memory for access from another port. If the referenced portion of
130279 typed memory was allocated discontiguously, the length thus determined may be
130280 shorter than anticipated, and the user code must adapt to the value returned.

130281 — Deallocation

130282 When a previously mapped subarea of typed memory is no longer mapped by any
130283 process in the system—as a result of a call or calls to *munmap*()—that subarea
130284 becomes potentially reusable for dynamic allocation; actual reuse of the subarea is a
130285 function of the dynamic typed memory allocation policy.

130286 — Unallocated Mapping

130287 It must be possible to map user-selected subareas of a typed memory object without
130288 marking that subarea as unavailable for allocation. This option is not the default
130289 behavior, and requires appropriate privileges.

130290 • Scenario

130291 The following scenario will serve to clarify the use of the typed memory interfaces.

130292 Process A running on P_1 (see Figure B-1, on page 3777) wants to allocate some memory
130293 from memory pool M_2, and it wants to share this portion of memory with process B
130294 running on P_2. Since P_2 only has access to the lower part of M_2, both processes will use the
130295 memory pool named M_{2b} which is the part of M_2 that is accessible both from P_1 and P_2. The
130296 operations that both processes need to perform are shown below:

130297 — Allocating Typed Memory

130298 Process A calls *posix_typed_mem_open*() with the name **/typed.m2b-b1** and a *tflag* of
130299 POSIX_TYPED_MEM_ALLOCATE to get a file descriptor usable for allocating from
130300 pool M_{2b} accessed through port B_1. It then calls *mmap*() with this file descriptor
130301 requesting a length of 4 096 bytes. The system allocates two discontiguous blocks of
130302 sizes 1 024 and 3 072 bytes within M_{2b}. The *mmap*() function returns a pointer to a
130303 4 096-byte array in process A's logical address space, mapping the allocated blocks
130304 contiguously. Process A can then utilize the array, and store data in it.

130305 — Determining the Location of the Allocated Blocks

130306 Process A can determine the lengths and offsets (relative to M_{2b}) of the two blocks
130307 allocated, by using the following procedure: First, process A calls *posix_mem_offset*( )
130308 with the address of the first element of the array and length 4 096. Upon return, the
130309 offset and length (1 024 bytes) of the first block are returned. A second call to
130310 *posix_mem_offset*( ) is then made using the address of the first element of the array
130311 plus 1 024 (the length of the first block), and a new length of 4 096–1 024. If there were
130312 more fragments allocated, this procedure could have been continued within a loop
130313 until the offsets and lengths of all the blocks were obtained. Notice that this relatively
130314 complex procedure can be avoided if contiguous allocation is requested (by opening
130315 the typed memory object with the *tflag*
130316 POSIX_TYPED_MEM_ALLOCATE_CONTIG).

130317 — Sharing Data Across Processes

130318 Process A passes the two offset values and lengths obtained from the
130319 *posix_mem_offset*( ) calls to process B running on P_2, via some form of interprocess
130320 communication. Process B can gain access to process A's data by calling
130321 *posix_typed_mem_open*( ) with the name **/typed.m2b-b2** and a *tflag* of zero, then using
130322 two *mmap*( ) calls on the resulting file descriptor to map the two subareas of that
130323 typed memory object to its own address space.

130324 • Rationale for no *mem_alloc*( ) and *mem_free*( )

130325 The standard developers had originally proposed a pair of new flags to *mmap*( ) which,
130326 when applied to a typed memory object descriptor, would cause *mmap*( ) to allocate
130327 dynamically from an unallocated and unmapped area of the typed memory object.
130328 Deallocation was similarly accomplished through the use of *munmap*( ). This was rejected
130329 by the ballot group because it excessively complicated the (already rather complex)
130330 *mmap*( ) interface and introduced semantics useful only for typed memory, to a function
130331 which must also map shared memory and files. They felt that a memory allocator should
130332 be built on top of *mmap*( ) instead of being incorporated within the same interface, much as
130333 the ISO C standard libraries build *malloc*( ) on top of the virtual memory mapping
130334 functions *brk*( ) and *sbrk*( ). This would eliminate the complicated semantics involved with
130335 unmapping only part of an allocated block of typed memory.

130336 To attempt to achieve ballot group consensus, typed memory allocation and deallocation
130337 was first migrated from *mmap*( ) and *munmap*( ) to a pair of complementary functions
130338 modeled on the ISO C standard *malloc*( ) and *free*( ). The *mem_alloc*( ) function specified
130339 explicitly the typed memory object (typed memory pool/access port) from which
130340 allocation takes place, unlike *malloc*( ) where the memory pool and port are unspecified.
130341 The *mem_free*( ) function handled deallocation. These new semantics still met all of the
130342 requirements detailed above without modifying the behavior of *mmap*( ) except to allow it
130343 to map specified areas of typed memory objects. An implementation would have been free
130344 to implement *mem_alloc*( ) and *mem_free*( ) over *mmap*( ), through *mmap*( ), or independently
130345 but cooperating with *mmap*( ).

130346 The ballot group was queried to see if this was an acceptable alternative, and while there
130347 was some agreement that it achieved the goal of removing the complicated semantics of
130348 allocation from the *mmap*( ) interface, several balloters realized that it just created two
130349 additional functions that behaved, in great part, like *mmap*( ). These balloters proposed an
130350 alternative which has been implemented here in place of a separate *mem_alloc*( ) and
130351 *mem_free*( ). This alternative is based on four specific suggestions:

1. The *posix_typed_mem_open*() function should provide a flag which specifies ``allocate on *mmap*()'' (otherwise, *mmap*() just maps the underlying object). This allows things roughly similar to **/dev/zero** *versus* **/dev/swap**. Two such flags have been implemented, one of which forces contiguous allocation.

2. The *posix_mem_offset*() function is acceptable because it can be applied usefully to mapped objects in general. It should return the file descriptor of the underlying object.

3. The *mem_get_info*() function in an earlier draft should be renamed *posix_typed_mem_get_info*() because it is not generally applicable to memory objects. It should probably return the file descriptor's allocation attribute. The renaming of the function has been implemented, but having it return a piece of information which is readily known by an application without this function has been rejected. Its whole purpose is to query the typed memory object for attributes that are not user-specified, but determined by the implementation.

4. There should be no separate *mem_alloc*() or *mem_free*() functions. Instead, using *mmap*() on a typed memory object opened with an ``allocate on *mmap*()'' flag should be used to force allocation. These are precisely the semantics defined in the current draft.

- Rationale for no Typed Memory Access Management

  The working group had originally defined an additional interface (and an additional kind of object: typed memory manager) to establish and dissolve mappings to typed memory on behalf of devices or processors which were independent of the operating system and had no inherent capability to directly establish mappings on their own. This was to have provided functionality similar to device driver interfaces such as *physio*() and their underlying bus-specific interfaces (for example, *mballoc*()) which serve to set up and break down DMA pathways, and derive mapped addresses for use by hardware devices and processor cards.

  The ballot group felt that this was beyond the scope of POSIX.1 and its amendments. Furthermore, the removal of interrupt handling interfaces from a preceding amendment (the IEEE Std 1003.1d-1999) during its balloting process renders these typed memory access management interfaces an incomplete solution to portable device management from a user process; it would be possible to initiate a device transfer to/from typed memory, but impossible to handle the transfer-complete interrupt in a portable way.

  To achieve ballot group consensus, all references to typed memory access management capabilities were removed. The concept of portable interfaces from a device driver to both operating system and hardware is being addressed by the Uniform Driver Interface (UDI) industry forum, with formal standardization deferred until proof of concept and industry-wide acceptance and implementation.

B.2.8.4 Process Scheduling

IEEE PASC Interpretation 1003.1 #96 has been applied, adding the *pthread_setschedprio*() function. This was added since previously there was no way for a thread to lower its own priority without going to the tail of the threads list for its new priority. This capability is necessary to bound the duration of priority inversion encountered by a thread.

The following portion of the rationale presents models, requirements, and standardization issues relevant to process and thread scheduling; see Section B.2.9.4 (on page 3824) for additional rationale relevant to thread scheduling.

In an operating system supporting multiple concurrent processes or threads, the system determines the order in which processes or threads execute to meet implementation-defined goals. For time-sharing systems, the goal is to enhance system throughput and promote fairness; the application is provided with little or no control over this sequencing function. While this is acceptable and desirable behavior in a time-sharing system, it is inappropriate in a realtime system; realtime applications must specifically control the execution sequence of their concurrent processes or threads in order to meet externally defined response requirements.

In POSIX.1-2024, the control over process and thread sequencing is provided using a concept of scheduling policies. These policies, described in detail in this section, define the behavior of the system whenever processor resources are to be allocated to competing processes or threads. Only the behavior of the policy is defined; conforming implementations are free to use any mechanism desired to achieve the described behavior.

- Models

  In an operating system supporting multiple concurrent processes or threads, the system determines the order in which threads (including those that are the only thread in a single-threaded process) execute and might force long-running threads to yield to other threads at certain intervals. Typically, the scheduling code is executed whenever an event occurs that might alter the thread to be executed next.

  The simplest scheduling strategy is a ``first-in, first-out'' (FIFO) dispatcher. Whenever a thread becomes runnable, it is placed on the end of a ready list. When processing resources become available, the thread at the front of the ready list starts or resumes execution and is removed from the list. This thread is executed until it exits or becomes blocked, at which point the processing resources used to execute it become available to execute another runnable thread. This scheduling technique is also known as ``run-to-completion'' or ``run-to-block''.

  A natural extension to this scheduling technique is the assignment of a ``non-migrating priority'' to each thread. This policy differs from strict FIFO scheduling in only one respect: whenever a thread becomes runnable, it is placed at the end of the list of threads runnable at that priority level. When selecting a thread to run, the system always selects the first thread from the highest priority queue with a runnable thread. Thus, when a thread becomes unblocked, it will preempt a running thread of lower priority without otherwise altering the ready list. Further, if a running or runnable thread's priority is altered, it is removed from the ready list for its old priority (if present in the list; that is, not running) and is inserted into the ready list for its new priority, according to the policy above, except that threads executing at a temporarily elevated priority as a consequence of owning a mutex initialized with the PTHREAD_PRIO_INHERIT or PTHREAD_PRIO_PROTECT protocol are exempted from this in order to ensure that a thread can lock and unlock such as mutex without the implicit yield that any resulting priority changes would normally cause.

  While the above policy might be considered unfriendly in a time-sharing environment in which multiple users require more balanced resource allocation, it could be ideal in a realtime environment for several reasons. The most important of these is that it is deterministic: the highest-priority thread is always run and, among threads of equal priority, the thread that has been runnable for the longest time is executed first. Because of this determinism, cooperating threads can implement more complex scheduling simply by altering their priority. For instance, if threads at a single priority were to reschedule themselves at fixed time intervals, a time-slice policy would result.

  In a dedicated operating system in which all threads belong to well-behaved realtime applications, non-migrating priority scheduling is sufficient. However, many existing

implementations provide for more complex scheduling policies.

For process scheduling, POSIX.1-2024 specifies a linear scheduling model. In this model, every process in the system has a priority. The system scheduler always dispatches a process that has the highest (generally the most time-critical) priority among all runnable processes in the system. As long as there is only one such process, the dispatching policy is trivial. When multiple processes of equal priority are eligible to run, they are ordered according to a strict run-to-completion (FIFO) policy. Thread scheduling is similar, except that the scheduling policy can be applied just to the threads within one process (PTHREAD_SCOPE_PROCESS scheduling contention scope) or to all threads system-wide (PTHREAD_SCOPE_SYSTEM scheduling contention scope). This and other considerations specific to thread scheduling are the subject of Section B.2.9.4 (on page 3824); the remainder of this section is described in terms of process scheduling but is also relevant to thread scheduling when read in conjunction with Section B.2.9.4.

The priority is represented as a positive integer and is inherited from the parent process. For processes running under a fixed priority scheduling policy, the priority is never altered except by an explicit function call.

It was determined arbitrarily that larger integers correspond to ``higher priorities''.

Certain implementations might impose restrictions on the priority ranges to which processes can be assigned. There also can be restrictions on the set of policies to which processes can be set.

- Requirements

Realtime processes require that scheduling be fast and deterministic, and that it guarantees to preempt lower priority processes.

Thus, given the linear scheduling model, realtime processes require that they be run at a priority that is higher than other processes. Within this framework, realtime processes are free to yield execution resources to each other in a completely portable and implementation-defined manner.

As there is a generally perceived requirement for processes at the same priority level to share processor resources more equitably, provisions are made by providing a scheduling policy (that is, SCHED_RR) intended to provide a timeslice-like facility.

Note: The following topics assume that low numeric priority implies low scheduling criticality and *vice versa*.

- Rationale for New Interface

Realtime applications need to be able to determine when processes will run in relation to each other. It must be possible to guarantee that a critical process will run whenever it is runnable; that is, whenever it wants to for as long as it needs. SCHED_FIFO satisfies this requirement. Additionally, SCHED_RR was defined to meet a realtime requirement for a well-defined time-sharing policy for processes at the same priority.

It would be possible to use the BSD *setpriority*( ) and *getpriority*( ) functions by redefining the meaning of the ``nice'' parameter according to the scheduling policy currently in use by the process. The System V *nice*( ) interface was felt to be undesirable for realtime because it specifies an adjustment to the ``nice'' value, rather than setting it to an explicit value. Realtime applications will usually want to set priority to an explicit value. Also, System V *nice*( ) does not allow for changing the priority of another process.

With the POSIX.1b interfaces, the traditional ``nice'' value does not affect the SCHED_FIFO or SCHED_RR scheduling policies. If a ``nice'' value is supported, it is implementation-defined whether it affects the SCHED_OTHER policy.

An important aspect of POSIX.1-2024 is the explicit description of the queuing and preemption rules. It is critical, to achieve deterministic scheduling, that such rules be stated clearly in POSIX.1-2024.

POSIX.1-2024 does not address the interaction between priority and swapping. The issues involved with swapping and virtual memory paging are extremely implementation-defined and would be nearly impossible to standardize at this point. The proposed scheduling paradigm, however, fully describes the scheduling behavior of runnable processes, of which one criterion is that the working set be resident in memory. Assuming the existence of a portable interface for locking portions of a process in memory, paging behavior need not affect the scheduling of realtime processes.

POSIX.1-2024 also does not address the priorities of ``system'' processes. In general, these processes should always execute in low-priority ranges to avoid conflict with other realtime processes. Implementations should document the priority ranges in which system processes run.

The default scheduling policy is not defined. The effect of I/O interrupts and other system processing activities is not defined. The temporary lending of priority from one process to another (such as for the purposes of affecting freeing resources) by the system is not addressed. Preemption of resources is not addressed. Restrictions on the ability of a process to affect other processes beyond a certain level (influence levels) is not addressed.

The rationale used to justify the simple time-quantum scheduler is that it is common practice to depend upon this type of scheduling to ensure ``fair'' distribution of processor resources among portions of the application that must interoperate in a serial fashion. Note that POSIX.1-2024 is silent with respect to the setting of this time quantum, or whether it is a system-wide value or a per-process value, although it appears that the prevailing realtime practice is for it to be a system-wide value.

In a system with N processes at a given priority, all processor-bound, in which the time quantum is equal for all processes at a specific priority level, the following assumptions are made of such a scheduling policy:

1. A time quantum Q exists and the current process will own control of the processor for at least a duration of Q and will have the processor for a duration of Q.
2. The Nth process at that priority will control a processor within a duration of $(N-1) \times Q$.

These assumptions are necessary to provide equal access to the processor and bounded response from the application.

The assumptions hold for the described scheduling policy only if no system overhead, such as interrupt servicing, is present. If the interrupt servicing load is non-zero, then one of the two assumptions becomes fallacious, based upon how Q is measured by the system.

If Q is measured by clock time, then the assumption that the process obtains a duration Q processor time is false if interrupt overhead exists. Indeed, a scenario can be constructed with N processes in which a single process undergoes complete processor starvation if a peripheral device, such as an analog-to-digital converter, generates significant interrupt activity periodically with a period of $N \times Q$.

If Q is measured as actual processor time, then the assumption that the Nth process runs in within the duration $(N-1) \times Q$ is false.

It should be noted that SCHED_FIFO suffers from interrupt-based delay as well. However, for SCHED_FIFO, the implied response of the system is ``as soon as possible'', so that the interrupt load for this case is a vendor selection and not a compliance issue.

With this in mind, it is necessary either to complete the definition by including bounds on the interrupt load, or to modify the assumptions that can be made about the scheduling policy.

Since the motivation of inclusion of the policy is common usage, and since current applications do not enjoy the luxury of bounded interrupt load, item (2) above is sufficient to express existing application needs and is less restrictive in the standard definition. No difference in interface is necessary.

In an implementation in which the time quantum is equal for all processes at a specific priority, our assumptions can then be restated as:

— A time quantum Q exists, and a processor-bound process will be rescheduled after a duration of, at most, Q. Time quantum Q may be defined in either wall clock time or execution time.

— In general, the Nth process of a priority level should wait no longer than $(N-1) \times Q$ time to execute, assuming no processes exist at higher priority levels.

— No process should wait indefinitely.

For implementations supporting per-process time quanta, these assumptions can be readily extended.

Austin Group Defect 1302 is applied, making requirements on *sched_yield*() also apply to *thrd_yield*().

Austin Group Defect 1610 is applied, clarifying the effects of PTHREAD_PRIO_INHERIT and PTHREAD_PRIO_PROTECT on scheduling queues.

Sporadic Server Scheduling Policy

The sporadic server is a mechanism defined for scheduling aperiodic activities in time-critical realtime systems. This mechanism reserves a certain bounded amount of execution capacity for processing aperiodic events at a high priority level. Any aperiodic events that cannot be processed within the bounded amount of execution capacity are executed in the background at a low priority level. Thus, a certain amount of execution capacity can be guaranteed to be available for processing periodic tasks, even under burst conditions in the arrival of aperiodic processing requests (that is, a large number of requests in a short time interval). The sporadic server also simplifies the schedulability analysis of the realtime system, because it allows aperiodic processes or threads to be treated as if they were periodic. The sporadic server was first described by Sprunt, et al.

The key concept of the sporadic server is to provide and limit a certain amount of computation capacity for processing aperiodic events at their assigned normal priority, during a time interval called the ``replenishment period''. Once the entity controlled by the sporadic server mechanism is initialized with its period and execution-time budget attributes, it preserves its execution capacity until an aperiodic request arrives. The request will be serviced (if there are no higher priority activities pending) as long as there is execution capacity left. If the request is completed, the actual execution time used to service it is subtracted from the capacity, and a replenishment of this amount of execution time is scheduled to happen one replenishment period after the arrival of the aperiodic request. If the request is not completed, because there is no execution capacity left, then the aperiodic process or thread is assigned a lower background priority. For each portion of consumed execution capacity the execution time used is replenished after one replenishment period. At the time of replenishment, if the sporadic server was executing at a background priority level, its priority is elevated to the normal level. Other similar replenishment policies have been defined, but the one presented here represents a compromise between efficiency and implementation complexity.

130588 The interface that appears in this section defines a new scheduling policy for threads and
130589 processes that behaves according to the rules of the sporadic server mechanism. Scheduling
130590 attributes are defined and functions are provided to allow the user to set and get the parameters
130591 that control the scheduling behavior of this mechanism, namely the normal and low priority, the
130592 replenishment period, the maximum number of pending replenishment operations, and the
130593 initial execution-time budget.

130594 • Scheduling Aperiodic Activities

130595 Virtually all realtime applications are required to process aperiodic activities. In many
130596 cases, there are tight timing constraints that the response to the aperiodic events must
130597 meet. Usual timing requirements imposed on the response to these events are:

130598 — The effects of an aperiodic activity on the response time of lower priority activities
130599 must be controllable and predictable.

130600 — The system must provide the fastest possible response time to aperiodic events.

130601 — It must be possible to take advantage of all the available processing bandwidth not
130602 needed by time-critical activities to enhance average-case response times to aperiodic
130603 events.

130604 Traditional methods for scheduling aperiodic activities are background processing, polling
130605 tasks, and direct event execution:

130606 — Background processing consists of assigning a very low priority to the processing of
130607 aperiodic events. It utilizes all the available bandwidth in the system that has not
130608 been consumed by higher priority threads. However, it is very difficult, or
130609 impossible, to meet requirements on average-case response time, because the
130610 aperiodic entity has to wait for the execution of all other entities which have higher
130611 priority.

130612 — Polling consists of creating a periodic process or thread for servicing aperiodic
130613 requests. At regular intervals, the polling entity is started and its services
130614 accumulated pending aperiodic requests. If no aperiodic requests are pending, the
130615 polling entity suspends itself until its next period. Polling allows the aperiodic
130616 requests to be processed at a higher priority level. However, worst and average-case
130617 response times of polling entities are a direct function of the polling period, and there
130618 is execution overhead for each polling period, even if no event has arrived. If the
130619 deadline of the aperiodic activity is short compared to the inter-arrival time, the
130620 polling frequency must be increased to guarantee meeting the deadline. For this case,
130621 the increase in frequency can dramatically reduce the efficiency of the system and,
130622 therefore, its capacity to meet all deadlines. Yet, polling represents a good way to
130623 handle a large class of practical problems because it preserves system predictability,
130624 and because the amortized overhead drops as load increases.

130625 — Direct event execution consists of executing the aperiodic events at a high fixed-
130626 priority level. Typically, the aperiodic event is processed by an interrupt service
130627 routine as soon as it arrives. This technique provides predictable response times for
130628 aperiodic events, but makes the response times of all lower priority activities
130629 completely unpredictable under burst arrival conditions. Therefore, if the density of
130630 aperiodic event arrivals is unbounded, it may be a dangerous technique for time-
130631 critical systems. Yet, for those cases in which the physics of the system imposes a
130632 bound on the event arrival rate, it is probably the most efficient technique.

130633 — The sporadic server scheduling algorithm combines the predictability of the polling
130634 approach with the short response times of the direct event execution. Thus, it allows
130635 systems to meet an important class of application requirements that cannot be met by

130636 using the traditional approaches. Multiple sporadic servers with different attributes
130637 can be applied to the scheduling of multiple classes of aperiodic events, each with
130638 different kinds of timing requirements, such as individual deadlines, average
130639 response times, and so on. It also has many other interesting applications for
130640 realtime, such as scheduling producer/consumer tasks in time-critical systems,
130641 limiting the effects of faults on the estimation of task execution-time requirements,
130642 and so on.

130643 • Existing Practice

130644 The sporadic server has been used in different kinds of applications, including military
130645 avionics, robot control systems, industrial automation systems, and so on. There are
130646 examples of many systems that cannot be successfully scheduled using the classic
130647 approaches, such as direct event execution, or polling, and are schedulable using a
130648 sporadic server scheduler. The sporadic server algorithm itself can successfully schedule
130649 all systems scheduled with direct event execution or polling.

130650 The sporadic server scheduling policy has been implemented as a commercial product in
130651 the run-time system of the Verdix Ada compiler. There are also many applications that
130652 have used a much less efficient application-level sporadic server. These realtime
130653 applications would benefit from a sporadic server scheduler implemented at the scheduler
130654 level.

130655 • Library-Level *versus* Kernel-Level Implementation

130656 The sporadic server interface described in this section requires the sporadic server policy
130657 to be implemented at the same level as the scheduler. This means that the process sporadic
130658 server must be implemented at the kernel level and the thread sporadic server policy
130659 implemented at the same level as the thread scheduler; that is, kernel or library level.

130660 In an earlier interface for the sporadic server, this mechanism was implementable at a
130661 different level than the scheduler. This feature allowed the implementor to choose between
130662 an efficient scheduler-level implementation, or a simpler user or library-level
130663 implementation. However, the working group considered that this interface made the use
130664 of sporadic servers more complex, and that library-level implementations would lack some
130665 of the important functionality of the sporadic server, namely the limitation of the actual
130666 execution time of aperiodic activities. The working group also felt that the interface
130667 described in this chapter does not preclude library-level implementations of threads
130668 intended to provide efficient low-overhead scheduling for those threads that are not
130669 scheduled under the sporadic server policy.

130670 • Range of Scheduling Priorities

130671 Each of the scheduling policies supported in POSIX.1-2024 has an associated range of
130672 priorities. The priority ranges for each policy might or might not overlap with the priority
130673 ranges of other policies. For time-critical realtime applications it is usual for periodic and
130674 aperiodic activities to be scheduled together in the same processor. Periodic activities will
130675 usually be scheduled using the SCHED_FIFO scheduling policy, while aperiodic activities
130676 may be scheduled using SCHED_SPORADIC. Since the application developer will require
130677 complete control over the relative priorities of these activities in order to meet his timing
130678 requirements, it would be desirable for the priority ranges of SCHED_FIFO and
130679 SCHED_SPORADIC to overlap completely. Therefore, although POSIX.1-2024 does not
130680 require any particular relationship between the different priority ranges, it is
130681 recommended that these two ranges should coincide.

- Dynamically Setting the Sporadic Server Policy

  Several members of the working group requested that implementations should not be required to support dynamically setting the sporadic server scheduling policy for a thread. The reason is that this policy may have a high overhead for library-level implementations of threads, and if threads are allowed to dynamically set this policy, this overhead can be experienced even if the thread does not use that policy. By disallowing the dynamic setting of the sporadic server scheduling policy, these implementations can accomplish efficient scheduling for threads using other policies. If a strictly conforming application needs to use the sporadic server policy, and is therefore willing to pay the overhead, it must set this policy at the time of thread creation.

- Limitation of the Number of Pending Replenishments

  The number of simultaneously pending replenishment operations must be limited for each sporadic server for two reasons: an unlimited number of replenishment operations would need an unlimited number of system resources to store all the pending replenishment operations; on the other hand, in some implementations each replenishment operation will represent a source of priority inversion (just for the duration of the replenishment operation) and thus, the maximum amount of replenishments must be bounded to guarantee bounded response times. The way in which the number of replenishments is bounded is by lowering the priority of the sporadic server to *sched_ss_low_priority* when the number of pending replenishments has reached its limit. In this way, no new replenishments are scheduled until the number of pending replenishments decreases.

  In the sporadic server scheduling policy defined in POSIX.1-2024, the application can specify the maximum number of pending replenishment operations for a single sporadic server, by setting the value of the *sched_ss_max_repl* scheduling parameter. This value must be between one and {SS_REPL_MAX}, which is a maximum limit imposed by the implementation. The limit {SS_REPL_MAX} must be greater than or equal to {_POSIX_SS_REPL_MAX}, which is defined to be four in POSIX.1-2024. The minimum limit of four was chosen so that an application can at least guarantee that four different aperiodic events can be processed during each interval of length equal to the replenishment period.

B.2.8.5 Clocks and Timers

- Clocks

  POSIX.1-2024 and the ISO C standard both define functions for obtaining system time. Implicit behind these functions is a mechanism for measuring passage of time. This specification makes this mechanism explicit and calls it a clock. The CLOCK_REALTIME clock required by POSIX.1-2024 is a higher resolution version of the clock that maintains POSIX.1 system time. This is a ``system-wide'' clock, in that it is visible to all processes and, were it possible for multiple processes to all read the clock at the same time, they would see the same value.

  An extensible interface was defined, with the ability for implementations to define additional clocks. This was done because of the observation that many realtime platforms support multiple clocks, and it was desired to fit this model within the standard interface. But implementation-defined clocks need not represent actual hardware devices, nor are they necessarily system-wide.

- Timers

  Two timer types are required for a system to support realtime applications:

130728 1. One-shot

130729 A one-shot timer is a timer that is armed with an initial expiration time, either
130730 relative to the current time or at an absolute time (based on some timing base, such
130731 as time in seconds and nanoseconds since the Epoch). The timer expires once and
130732 then is disarmed. With the specified facilities, this is accomplished by setting the
130733 *it_value* member of the *value* argument to the desired expiration time and the
130734 *it_interval* member to zero.

130735 2. Periodic

130736 A periodic timer is a timer that is armed with an initial expiration time, again either
130737 relative or absolute, and a repetition interval. When the initial expiration occurs,
130738 the timer is reloaded with the repetition interval and continues counting. With the
130739 specified facilities, this is accomplished by setting the *it_value* member of the *value*
130740 argument to the desired initial expiration time and the *it_interval* member to the
130741 desired repetition interval.

130742 For both of these types of timers, the time of the initial timer expiration can be specified in
130743 two ways:

130744 1. Relative (to the current time)

130745 2. Absolute

130746 • Examples of Using Realtime Timers

130747 In the diagrams below, *S* indicates a program schedule, *R* shows a schedule method
130748 request, and *E* suggests an internal operating system event.

130749 — Periodic Timer: Data Logging

130750 During an experiment, it might be necessary to log realtime data periodically to an
130751 internal buffer or to a mass storage device. With a periodic scheduling method, a
130752 logging module can be started automatically at fixed time intervals to log the data.

130753 Program schedule is requested every 10 seconds.

130754
```
    R          S          S          S          S          S
130755 ----+----+----+----+----+----+----+----+----+----+----+--->
130756     5   10   15   20   25   30   35   40   45   50   55
```

130757 [Time (in Seconds)]

130758 To achieve this type of scheduling using the specified facilities, one would allocate a
130759 per-process timer based on clock ID CLOCK_REALTIME. Then the timer would be
130760 armed via a call to *timer_settime*( ) with the TIMER_ABSTIME flag reset, and with an
130761 initial expiration value and a repetition interval of 10 seconds.

130762 — One-shot Timer (Relative Time): Device Initialization

130763 In an emission test environment, large sample bags are used to capture the exhaust
130764 from a vehicle. The exhaust is purged from these bags before each and every test.
130765 With a one-shot timer, a module could initiate the purge function and then suspend
130766 itself for a predetermined period of time while the sample bags are prepared.

130767 Program schedule requested 20 seconds after call is issued.

130768
```
    R                    S
130769 ----+----+----+----+----+----+----+----+----+----+----+--->
130770     5   10   15   20   25   30   35   40   45   50   55
```


[Time (in Seconds)]

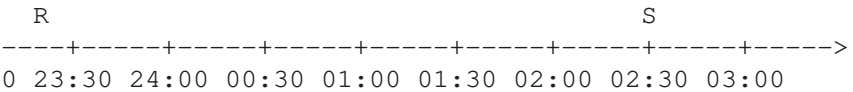
To achieve this type of scheduling using the specified facilities, one would allocate a per-process timer based on clock ID CLOCK_REALTIME. Then the timer would be armed via a call to *timer_settime*() with the TIMER_ABSTIME flag reset, and with an initial expiration value of 20 seconds and a repetition interval of zero.

Note that if the program wishes merely to suspend itself for the specified interval, it could more easily use *nanosleep*().

— One-shot Timer (Absolute Time): Data Transmission

The results from an experiment are often moved to a different system within a network for post-processing or archiving. With an absolute one-shot timer, a module that moves data from a test-cell computer to a host computer can be automatically scheduled on a daily basis.

Program schedule requested for 2:30 a.m.



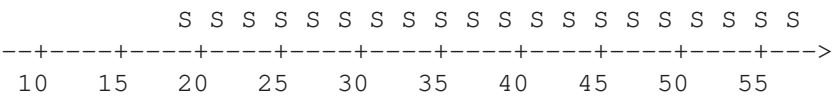
[Time of Day]

To achieve this type of scheduling using the specified facilities, a per-process timer would be allocated based on clock ID CLOCK_REALTIME. Then the timer would be armed via a call to *timer_settime*() with the TIMER_ABSTIME flag set, and an initial expiration value equal to 2:30 a.m. of the next day.

— Periodic Timer (Relative Time): Signal Stabilization

Some measurement devices, such as emission analyzers, do not respond instantaneously to an introduced sample. With a periodic timer with a relative initial expiration time, a module that introduces a sample and records the average response could suspend itself for a predetermined period of time while the signal is stabilized and then sample at a fixed rate.

Program schedule requested 15 seconds after call is issued and every 2 seconds thereafter.



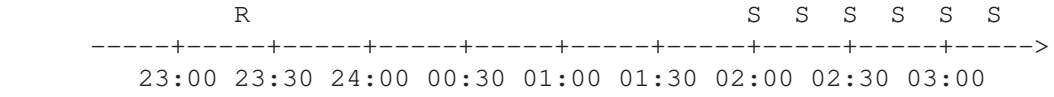
[Time (in Seconds)]

To achieve this type of scheduling using the specified facilities, one would allocate a per-process timer based on clock ID CLOCK_REALTIME. Then the timer would be armed via a call to *timer_settime*() with TIMER_ABSTIME flag reset, and with an initial expiration value of 15 seconds and a repetition interval of 2 seconds.

— Periodic Timer (Absolute Time): Work Shift-related Processing

Resource utilization data is useful when time to perform experiments is being scheduled at a facility. With a periodic timer with an absolute initial expiration time, a module can be scheduled at the beginning of a work shift to gather resource utilization data throughout the shift. This data can be used to allocate resources effectively to minimize bottlenecks and delays and maximize facility throughput.

Program schedule requested for 2:00 a.m. and every 15 minutes thereafter.



To achieve this type of scheduling using the specified facilities, one would allocate a per-process timer based on clock ID CLOCK_REALTIME. Then the timer would be armed via a call to *timer_settime*( ) with TIMER_ABSTIME flag set, and with an initial expiration value equal to 2:00 a.m. and a repetition interval equal to 15 minutes.

- Relationship of Timers to Clocks

The relationship between clocks and timers armed with an absolute time is straightforward: a timer expiration signal is requested when the associated clock reaches or exceeds the specified time. The relationship between clocks and timers armed with a relative time (an interval) is less obvious, but not unintuitive. In this case, a timer expiration signal is requested when the specified interval, *as measured by the associated clock*, has passed. For the required CLOCK_REALTIME clock, this allows timer expiration signals to be requested at specified ``wall clock'' times (absolute), or when a specified interval of ``realtime'' has passed (relative). For an implementation-defined clock—say, a process virtual time clock—timer expirations could be requested when the process has used a specified total amount of virtual time (absolute), or when it has used a specified *additional* amount of virtual time (relative).

The interfaces also allow flexibility in the implementation of the functions. For example, an implementation could convert all absolute times to intervals by subtracting the clock value at the time of the call from the requested expiration time and ``counting down'' at the supported resolution. Or it could convert all relative times to absolute expiration time by adding in the clock value at the time of the call and comparing the clock value to the expiration time at the supported resolution. Or it might even choose to maintain absolute times as absolute and compare them to the clock value at the supported resolution for absolute timers, and maintain relative times as intervals and count them down at the resolution supported for relative timers. The choice will be driven by efficiency considerations and the underlying hardware or software clock implementation.

- Data Definitions for Clocks and Timers

POSIX.1-2024 uses a time representation capable of supporting nanosecond resolution timers for the following reasons:

— To enable POSIX.1-2024 to represent those computer systems already using nanosecond or submicrosecond resolution clocks.

— To accommodate those per-process timers that might need nanoseconds to specify an absolute value of system-wide clocks, even though the resolution of the per-process timer may only be milliseconds, or *vice versa*.

— Because the number of nanoseconds in a second can be represented in 32 bits.

Time values are represented in the **timespec** structure. The *tv_sec* member is of type **time_t** so that this member is compatible with time values used by POSIX.1 functions and the ISO C standard. The *tv_nsec* member is a **signed long** in order to simplify and clarify code that decrements or finds differences of time values. Note that because 1 billion (number of nanoseconds per second) is less than half of the value representable by a signed 32-bit value, it is always possible to add two valid fractional seconds represented as integral nanoseconds without overflowing the signed 32-bit value.

A maximum allowable resolution for the CLOCK_REALTIME clock of 20 ms (1/50

seconds) was chosen to allow line frequency clocks in European countries to be conforming. 60 Hz clocks in the US will also be conforming, as will finer granularity clocks, although a Strictly Conforming Application cannot assume a granularity of less than 20 ms (1/50 seconds).

The minimum allowable maximum time allowed for the CLOCK_REALTIME clock and the function *nanosleep*(), and timers created with *clock_id*=CLOCK_REALTIME, is determined by the fact that the *tv_sec* member is of type **time_t**.

POSIX.1-2024 specifies that timer expirations must not be delivered early, and *nanosleep*() must not return early due to quantization error. POSIX.1-2024 discusses the various implementations of *alarm*() in the rationale and states that implementations that do not allow alarm signals to occur early are the most appropriate, but refrained from mandating this behavior. Because of the importance of predictability to realtime applications, POSIX.1-2024 takes a stronger stance.

The standard developers considered using a time representation that differs from POSIX.1b in the second 32 bit of the 64-bit value. Whereas POSIX.1b defines this field as a fractional second in nanoseconds, the other methodology defines this as a binary fraction of one second, with the radix point assumed before the most significant bit.

POSIX.1b is a software, source-level standard and most of the benefits of the alternate representation are enjoyed by hardware implementations of clocks and algorithms. It was felt that mandating this format for POSIX.1b clocks and timers would unnecessarily burden the application developer with writing, possibly non-portable, multiple precision arithmetic packages to perform conversion between binary fractions and integral units such as nanoseconds, milliseconds, and so on.

Rationale for the Monotonic Clock

For those applications that use time services to achieve realtime behavior, changing the value of the clock on which these services rely may cause erroneous timing behavior. For these applications, it is necessary to have a monotonic clock which cannot run backwards, and which has a maximum clock jump that is required to be documented by the implementation. Additionally, it is desirable (but not required by POSIX.1-2024) that the monotonic clock increases its value uniformly. This clock should not be affected by changes to the system time; for example, to synchronize the clock with an external source or to account for leap seconds. Such changes would cause errors in the measurement of time intervals for those time services that use the absolute value of the clock.

One could argue that by defining the behavior of time services when the value of a clock is changed, deterministic realtime behavior can be achieved. For example, one could specify that relative time services should be unaffected by changes in the value of a clock. However, there are time services that are based upon an absolute time, but that are essentially intended as relative time services. For example, *pthread_cond_timedwait*() uses an absolute time to allow it to wake up after the required interval despite spurious wakeups. Although sometimes the *pthread_cond_timedwait*() timeouts are absolute in nature, there are many occasions in which they are relative, and their absolute value is determined from the current time plus a relative time interval. In this latter case, if the clock changes while the thread is waiting, the wait interval will not be the expected length. If a *pthread_cond_timedwait*() function were created that would take a relative time, it would not solve the problem because to retain the intended ``deadline'' a thread would need to compensate for latency due to the spurious wakeup, and preemption between wakeup and the next wait.

The solution is to create a new monotonic clock, whose value does not change except for the regular ticking of the clock, and use this clock for implementing the various relative timeouts

that appear in the different POSIX interfaces, as well as allow *pthread_cond_timedwait*( ) to choose this new clock for its timeout. A new *clock_nanosleep*( ) function is created to allow an application to take advantage of this newly defined clock. Notice that the monotonic clock may be implemented using the same hardware clock as the system clock.

Relative timeouts for *sigtimedwait*( ) and *aio_suspend*( ) have been redefined to use the monotonic clock, if present. The *alarm*( ) function has not been redefined, because the same effect but with better resolution can be achieved by creating a timer (for which the appropriate clock may be chosen).

The *pthread_cond_timedwait*( ) function has been treated in a different way, compared to other functions with absolute timeouts, because it is used to wait for an event, and thus it may have a deadline, while the other timeouts are generally used as an error recovery mechanism, and for them the use of the monotonic clock is not so important. Since the desired timeout for the *pthread_cond_timedwait*( ) function may either be a relative interval or an absolute time of day deadline, a new initialization attribute has been created for condition variables to specify the clock that is used for measuring the timeout in a call to *pthread_cond_timedwait*( ). In this way, if a relative timeout is desired, the monotonic clock will be used; if an absolute deadline is required instead, the CLOCK_REALTIME or another appropriate clock may be used. For condition variables, this capability is also available by passing CLOCK_MONOTONIC to the *pthread_cond_clockwait*( ) function. Similarly, CLOCK_MONOTONIC can be specified when calling *pthread_mutex_clocklock*( ), *pthread_rwlock_clockrdlock*( ), *pthread_rwlock_clockwrlock*( ), and *sem_clockwait*( ).

It was later found necessary to add variants of almost all interfaces that accept absolute timeouts that allow the clock to be specified. This is because, despite the claim in the previous paragraph, it is not possible to safely use a CLOCK_REALTIME absolute timeout even to prevent errors when the system clock is warped by a potentially large amount. A ``safety timeout'' of a minute on a call to *pthread_mutex_timedlock*( ) could actually mean that the call would return ETIMEDOUT early without acquiring the lock if the system clock is warped forwards immediately prior to or during the call. On the other hand, a short timeout could end up being arbitrarily long if the system clock is warped backwards immediately prior to or during the call. These problems are solved by the new *clockwait* and *clocklock* variants of the existing *timedwait* and *timedlock* functions. These variants accept an extra **clockid_t** parameter to indicate the clock to be used for the wait. The clock ID is passed rather than using attributes as previously for *pthread_cond_timedwait*( ) in order to allow the ISO/IEC 14882:2011 standard (C++11) and later to be implemented correctly. C++ requires that the clock to use for the wait is not known until the time of the wait call, so it cannot be supplied during creation. The new functions are *pthread_cond_clockwait*( ), *pthread_mutex_clocklock*( ), *pthread_mutex_clockrdlock*( ), *pthread_mutex_clockwrlock*( ), and *sem_clockwait*( ). It is expected that *mq_clockreceive( )* and *mq_clocksend( )* functions will be added in a future version of this standard.

The *nanosleep*( ) function has not been modified with the introduction of the monotonic clock. Instead, a new *clock_nanosleep*( ) function has been created, in which the desired clock may be specified in the function call.

- History of Resolution Issues

  Due to the shift from relative to absolute timeouts in IEEE Std 1003.1d-1999, the amendments to the *sem_timedwait*( ), *pthread_mutex_timedlock*( ), *mq_timedreceive*( ), and *mq_timedsend*( ) functions of that standard have been removed. Those amendments specified that CLOCK_MONOTONIC would be used for the (relative) timeouts if the (optional at the time) Monotonic Clock was supported.

  Having these functions continue to be tied solely to CLOCK_MONOTONIC would not work. Since the absolute value of a time value obtained from CLOCK_MONOTONIC is

unspecified, under the absolute timeouts interface, applications would behave differently depending on whether the Monotonic Clock was supported or not (because the absolute value of the clock would have different meanings in either case).

Two options were considered:

1. Leave the current behavior unchanged, which specifies the CLOCK_REALTIME clock for these (absolute) timeouts, to allow portability of applications between implementations supporting or not the Monotonic Clock.
2. Modify these functions in the way that *pthread_cond_timedwait*( ) was modified to allow a choice of clock, so that an application could use CLOCK_REALTIME when it is trying to achieve an absolute timeout and CLOCK_MONOTONIC when it is trying to achieve a relative timeout.

It was decided that the features of CLOCK_MONOTONIC are not as critical to these functions as they are to *pthread_cond_timedwait*( ). The *pthread_cond_timedwait*( ) function is given an absolute timeout; the timeout may represent a deadline for an event. When other functions are given relative timeouts, the timeouts are typically for error recovery purposes and need not be so precise.

Therefore, it was decided that these functions should be tied to CLOCK_REALTIME and not complicated by being given a choice of clock.

Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.

Execution Time Monitoring

- Introduction

The main goals of the execution time monitoring facilities defined in this chapter are to measure the execution time of processes and threads and to allow an application to establish CPU time limits for these entities.

The analysis phase of time-critical realtime systems often relies on the measurement of execution times of individual threads or processes to determine whether the timing requirements will be met. Also, performance analysis techniques for soft deadline realtime systems rely heavily on the determination of these execution times. The execution time monitoring functions provide application developers with the ability to measure these execution times online and open the possibility of dynamic execution-time analysis and system reconfiguration, if required.

The second goal of allowing an application to establish execution time limits for individual processes or threads and detecting when they overrun allows program robustness to be increased by enabling online checking of the execution times.

If errors are detected—possibly because of erroneous program constructs, the existence of errors in the analysis phase, or a burst of event arrivals—online detection and recovery is possible in a portable way. This feature can be extremely important for many time-critical applications. Other applications require trapping CPU-time errors as a normal way to exit an algorithm; for instance, some realtime artificial intelligence applications trigger a number of independent inference processes of varying accuracy and speed, limit how long they can run, and pick the best answer available when time runs out. In many periodic systems, overrun processes are simply restarted in the next resource period, after necessary end-of-period actions have been taken. This allows algorithms that are inherently data-dependent to be made predictable.

The interface that appears in this chapter defines a new type of clock, the CPU-time clock, which measures execution time. Each process or thread can invoke the clock and timer

functions defined in POSIX.1 to use them. Functions are also provided to access the CPU-time clock of other processes or threads to enable remote monitoring of these clocks. Monitoring of threads of other processes is not supported, since these threads are not visible from outside of their own process with the interfaces defined in POSIX.1.

- Execution Time Monitoring Interface

The clock and timer interface defined in POSIX.1 historically only defined one clock, which measures wall-clock time. The requirements for measuring execution time of processes and threads, and setting limits to their execution time by detecting when they overrun, can be accomplished with that interface if a new kind of clock is defined. These new clocks measure execution time, and one is associated with each process and with each thread. The clock functions currently defined in POSIX.1 can be used to read and set these CPU-time clocks, and timers can be created using these clocks as their timing base. These timers can then be used to send a signal when some specified execution time has been exceeded. The CPU-time clocks of each process or thread can be accessed by using the symbols CLOCK_PROCESS_CPUTIME_ID or CLOCK_THREAD_CPUTIME_ID.

The clock and timer interface defined in POSIX.1 and extended with the new kind of CPU-time clock would only allow processes or threads to access their own CPU-time clocks. However, many realtime systems require the possibility of monitoring the execution time of processes or threads from independent monitoring entities. In order to allow applications to construct independent monitoring entities that do not require cooperation from or modification of the monitored entities, two functions have been added: *clock_getcpuclockid*(), for accessing CPU-time clocks of other processes, and *pthread_getcpuclockid*(), for accessing CPU-time clocks of other threads. These functions return the clock identifier associated with the process or thread specified in the call. These clock IDs can then be used in the rest of the clock function calls.

The clocks accessed through these functions could also be used as a timing base for the creation of timers, thereby allowing independent monitoring entities to limit the CPU time consumed by other entities. However, this possibility would imply additional complexity and overhead because of the need to maintain a timer queue for each process or thread, to store the different expiration times associated with timers created by different processes or threads. The working group decided this additional overhead was not justified by application requirements. Therefore, creation of timers attached to the CPU-time clocks of other processes or threads has been specified as implementation-defined.

- Overhead Considerations

The measurement of execution time may introduce additional overhead in the thread scheduling, because of the need to keep track of the time consumed by each of these entities. In library-level implementations of threads, the efficiency of scheduling could be somehow compromised because of the need to make a kernel call, at each context switch, to read the process CPU-time clock. Consequently, a thread creation attribute called *cpu-clock-requirement* was defined, to allow threads to disconnect their respective CPU-time clocks. However, the Ballot Group considered that this attribute itself introduced some overhead, and that in current implementations it was not worth the effort. Therefore, the attribute was deleted, and thus thread CPU-time clocks are required for all threads if the Thread CPU-Time Clocks option is supported.

- Accuracy of CPU-Time Clocks

The mechanism used to measure the execution time of processes and threads is specified in POSIX.1-2024 as implementation-defined. The reason for this is that both the underlying hardware and the implementation architecture have a very strong influence on the accuracy achievable for measuring CPU time. For some implementations, the specification

of strict accuracy requirements would represent very large overheads, or even the impossibility of being implemented.

Since the mechanism for measuring execution time is implementation-defined, realtime applications will be able to take advantage of accurate implementations using a portable interface. Of course, strictly conforming applications cannot rely on any particular degree of accuracy, in the same way as they cannot rely on a very accurate measurement of wall clock time. There will always exist applications whose accuracy or efficiency requirements on the implementation are more rigid than the values defined in POSIX.1-2024 or any other standard.

In any case, there is a minimum set of characteristics that realtime applications would expect from most implementations. One such characteristic is that the sum of all the execution times of all the threads in a process equals the process execution time, when no CPU-time clocks are disabled. This need not always be the case because implementations may differ in how they account for time during context switches. Another characteristic is that the sum of the execution times of all processes in a system equals the number of processors, multiplied by the elapsed time, assuming that no processor is idle during that elapsed time. However, in some implementations it might not be possible to relate CPU time to elapsed time. For example, in a heterogeneous multi-processor system in which each processor runs at a different speed, an implementation may choose to define each ``second'' of CPU time to be a certain number of ``cycles'' that a CPU has executed.

- Existing Practice

Measuring and limiting the execution time of each concurrent activity are common features of most industrial implementations of realtime systems. Almost all critical realtime systems are currently built upon a cyclic executive. With this approach, a regular timer interrupt kicks off the next sequence of computations. It also checks that the current sequence has completed. If it has not, then some error recovery action can be undertaken (or at least an overrun is avoided). Current software engineering principles and the increasing complexity of software are driving application developers to implement these systems on multi-threaded or multi-process operating systems. Therefore, if a POSIX operating system is to be used for this type of application, then it must offer the same level of protection.

Execution time clocks are also common in most UNIX implementations, although these clocks usually have requirements different from those of realtime applications. The POSIX.1 *times*() function supports the measurement of the execution time of the calling process, and its terminated child processes. This execution time is measured in clock ticks and is supplied as two different values with the user and system execution times, respectively. BSD supports the function *getrusage*(), which allows the calling process to get information about the resources used by itself and/or all of its terminated child processes. The resource usage includes user and system CPU time. Some UNIX systems have options to specify high resolution (up to one microsecond) CPU-time clocks using the *times*() or the *getrusage*() functions.

The *times*() and *getrusage*() interfaces do not meet important realtime requirements, such as the possibility of monitoring execution time from a different process or thread, or the possibility of detecting an execution time overrun. The latter requirement is supported in some UNIX implementations that are able to send a signal when the execution time of a process has exceeded some specified value. For example, BSD defines the functions *getitimer*() and *setitimer*(), which can operate either on a realtime clock (wall-clock), or on virtual-time or profile-time clocks which measure CPU time in two different ways. These functions do not support access to the execution time of other processes.

At least one operating system supports per-process and per-thread execution time clocks, and also supports limiting the execution time of a given process.

Given all this existing practice, the working group considered that the POSIX.1 clocks and timers interface was appropriate to meet most of the requirements that realtime applications have for execution time clocks. Functions were added to get the CPU time clock IDs, and to allow/disallow the thread CPU-time clocks (in order to preserve the efficiency of some implementations of threads).

- Clock Constants

The definition of the manifest constants CLOCK_PROCESS_CPUTIME_ID and CLOCK_THREAD_CPUTIME_ID allows processes or threads, respectively, to access their own execution-time clocks. However, given a process or thread, access to its own execution-time clock is also possible if the clock ID of this clock is obtained through a call to *clock_getcpuclockid*() or *pthread_getcpuclockid*(). Therefore, these constants are not necessary and could be deleted to make the interface simpler. Their existence saves one system call in the first access to the CPU-time clock of each process or thread. The working group considered this issue and decided to leave the constants in POSIX.1-2024 because they are closer to the POSIX.1b use of clock identifiers.

- Library Implementations of Threads

In library implementations of threads, kernel entities and library threads can coexist. In this case, if the CPU-time clocks are supported, most of the clock and timer functions will need to have two implementations: one in the thread library, and one in the system calls library. The main difference between these two implementations is that the thread library implementation will have to deal with clocks and timers that reside in the thread space, while the kernel implementation will operate on timers and clocks that reside in kernel space. In the library implementation, if the clock ID refers to a clock that resides in the kernel, a kernel call will have to be made. The correct version of the function can be chosen by specifying the appropriate order for the libraries during the link process.

- History of Resolution Issues: Deletion of the *enable* Attribute

In early proposals, consideration was given to inclusion of an attribute called *enable* for CPU-time clocks. This would allow implementations to avoid the overhead of measuring execution time for those processes or threads for which this measurement was not required. However, this is unnecessary since processes are already required to measure execution time by the POSIX.1 *times*() function. Consequently, the *enable* attribute is not present.

Rationale Relating to Timeouts

- Requirements for Timeouts

Realtime systems which must operate reliably over extended periods without human intervention are characteristic in embedded applications such as avionics, machine control, and space exploration, as well as more mundane applications such as cable TV, security systems, and plant automation. A multi-tasking paradigm, in which many independent and/or cooperating software functions relinquish the processor(s) while waiting for a specific stimulus, resource, condition, or operation completion, is very useful in producing well engineered programs for such systems. For such systems to be robust and fault-tolerant, expected occurrences that are unduly delayed or that never occur must be detected so that appropriate recovery actions may be taken. This is difficult if there is no way for a task to regain control of a processor once it has relinquished control (blocked) awaiting an occurrence which, perhaps because of corrupted code, hardware malfunction,

131150 or latent software bugs, will not happen when expected. Therefore, the common practice
131151 in realtime operating systems is to provide a capability to time out such blocking services.
131152 Although there are several methods to achieve this already defined by POSIX, none are as
131153 reliable or efficient as initiating a timeout simultaneously with initiating a blocking service.
131154 This is especially critical in hard-realtime embedded systems because the processors
131155 typically have little time reserve, and allowed fault recovery times are measured in
131156 milliseconds rather than seconds.

131157 The working group largely agreed that such timeouts were necessary and ought to become
131158 part of POSIX.1-2024, particularly vendors of realtime operating systems whose customers
131159 had already expressed a strong need for timeouts. There was some resistance to inclusion
131160 of timeouts in POSIX.1-2024 because the desired effect, fault tolerance, could, in theory, be
131161 achieved using existing facilities and alternative software designs, but there was no
131162 compelling evidence that realtime system designers would embrace such designs at the
131163 sacrifice of performance and/or simplicity.

131164 • Which Services should be Timed Out?

131165 Originally, the working group considered the prospect of providing timeouts on all
131166 blocking services, including those currently existing in POSIX.1, POSIX.1b, and POSIX.1c,
131167 and future interfaces to be defined by other working groups, as sort of a general policy.
131168 This was rather quickly rejected because of the scope of such a change, and the fact that
131169 many of those services would not normally be used in a realtime context. More traditional
131170 timesharing solutions to timeout would suffice for most of the POSIX.1 interfaces, while
131171 others had asynchronous alternatives which, while more complex to utilize, would be
131172 adequate for some realtime and all non-realtime applications.

131173 The list of potential candidates for timeouts was narrowed to the following for further
131174 consideration:

131175 — POSIX.1b

131176 — *sem_wait*( )

131177 — *mq_receive*( )

131178 — *mq_send*( )

131179 — *lio_listio*( )

131180 — *aio_suspend*( )

131181 — *sigwait*( ) (timeout already implemented by *sigtimedwait*( ))

131182 — POSIX.1c

131183 — *pthread_mutex_lock*( )

131184 — *pthread_join*( )

131185 — *pthread_cond_wait*( )
131186 (timeout already implemented by *pthread_cond_timedwait*( ))

131187 — POSIX.1

131188 — *read*( )

131189 — *write*( )

131190 After further review by the working group, the *lio_listio*( ), *read*( ), and *write*( ) functions (all
131191 forms of blocking synchronous I/O) were eliminated from the list because of the
131192 following:

— Asynchronous alternatives exist

— Timeouts can be implemented, albeit non-portably, in device drivers

— A strong desire not to introduce modifications to POSIX.1 interfaces

The working group ultimately rejected *pthread_join*() since both that interface and a timed variant of that interface are non-minimal and may be implemented as a function. See below for a library implementation of *pthread_join*().

Thus, there was a consensus among the working group members to add timeouts to 4 of the remaining 5 functions (the timeout for *aio_suspend*() was ultimately added directly to POSIX.1b, while the others were added by POSIX.1d). However, *pthread_mutex_lock*() remained contentious.

Many feel that *pthread_mutex_lock*() falls into the same class as the other functions; that is, it is desirable to time out a mutex lock because a mutex may fail to be unlocked due to errant or corrupted code in a critical section (looping or branching outside of the unlock code), and therefore is equally in need of a reliable, simple, and efficient timeout. In fact, since mutexes are intended to guard small critical sections, most *pthread_mutex_lock*() calls would be expected to obtain the lock without blocking nor utilizing any kernel service, even in implementations of threads with global contention scope; the timeout alternative need only be considered after it is determined that the thread must block.

Those opposed to timing out mutexes feel that the very simplicity of the mutex is compromised by adding a timeout semantic, and that to do so is senseless. They claim that if a timed mutex is really deemed useful by a particular application, then it can be constructed from the facilities already in POSIX.1b and POSIX.1c. The following two C-language library implementations of mutex locking with timeout represent the solutions offered (in both implementations, the timeout parameter is specified as absolute time, not relative time as in the proposed POSIX.1c interfaces).

- Spinlock Implementation

```
#include <pthread.h>
#include <time.h>
#include <errno.h>

int pthread_mutex_timedlock(pthread_mutex_t *mutex,
        const struct timespec *timeout)
{
    struct timespec timenow;

    while (pthread_mutex_trylock(mutex) == EBUSY)
    {
        clock_gettime(CLOCK_REALTIME, &timenow);
        if (timespec_cmp(&timenow,timeout) >= 0)
        {
            return ETIMEDOUT;
        }
        sched_yield();
    }
    return 0;
}
```

The Spinlock implementation is generally unsuitable for any application using priority-based thread scheduling policies such as SCHED_FIFO or SCHED_RR, since the mutex could currently be held by a thread of lower priority within the same allocation domain,

131240 but since the waiting thread never blocks, only threads of equal or higher priority will ever
131241 run, and the mutex cannot be unlocked. Setting priority inheritance or priority ceiling
131242 protocol on the mutex does not solve this problem, since the priority of a mutex owning
131243 thread is only boosted if higher priority threads are blocked waiting for the mutex; clearly
131244 not the case for this spinlock.

131245 • Condition Wait Implementation

```
131246 #include <pthread.h>
131247 #include <time.h>
131248 #include <errno.h>

131249 struct timed_mutex {
131250     int locked;
131251     pthread_mutex_t mutex;
131252     pthread_cond_t cond;
131253 };
131254 typedef struct timed_mutex timed_mutex_t;

131255 int timed_mutex_lock(timed_mutex_t *tm,
131256         const struct timespec *timeout)
131257 {
131258     int timedout=FALSE;
131259     int error_status;

131260     pthread_mutex_lock(&tm->mutex);

131261     while (tm->locked && !timedout)
131262     {
131263         if ((error_status=pthread_cond_timedwait(&tm->cond,
131264             &tm->mutex, timeout))!=0)
131265         {
131266             if (error_status==ETIMEDOUT) timedout = TRUE;
131267         }
131268     }

131269     if(timedout)
131270     {
131271         pthread_mutex_unlock(&tm->mutex);
131272         return ETIMEDOUT;
131273     }
131274     else
131275     {
131276         tm->locked = TRUE;
131277         pthread_mutex_unlock(&tm->mutex);
131278         return 0;
131279     }
131280 }

131281 void timed_mutex_unlock(timed_mutex_t *tm)
131282 {
131283     pthread_mutex_lock(&tm->mutex); / for case assignment not atomic /
131284     tm->locked = FALSE;
131285     pthread_mutex_unlock(&tm->mutex);
131286     pthread_cond_signal(&tm->cond);
131287 }
```


The Condition Wait implementation effectively substitutes the *pthread_cond_timedwait*() function (which is currently timed out) for the desired *pthread_mutex_timedlock*(). Since waits on condition variables currently do not include protocols which avoid priority inversion, this method is generally unsuitable for realtime applications because it does not provide the same priority inversion protection as the untimed *pthread_mutex_lock*(). Also, for any given implementations of the current mutex and condition variable primitives, this library implementation has a performance cost at least 2.5 times that of the untimed *pthread_mutex_lock*() even in the case where the timed mutex is readily locked without blocking (the interfaces required for this case are shown in bold). Even in uniprocessors or where assignment is atomic, at least an additional *pthread_cond_signal*() is required. *pthread_mutex_timedlock*() could be implemented at effectively no performance penalty in this case because the timeout parameters need only be considered after it is determined that the mutex cannot be locked immediately.

Thus it has not yet been shown that the full semantics of mutex locking with timeout can be efficiently and reliably achieved using existing interfaces. Even if the existence of an acceptable library implementation were proven, it is difficult to justify why the interface itself should not be made portable, especially considering approval for the other four timeouts.

- Rationale for Library Implementation of *pthread_timedjoin*()

Library implementation of *pthread_timedjoin*():

```
/*
 * Construct a thread variety entirely from existing functions
 * with which a join can be done, allowing the join to time out.
 */
#include <pthread.h>
#include <time.h>

struct timed_thread {
    pthread_t t;
    pthread_mutex_t m;
    int exiting;
    pthread_cond_t exit_c;
    void *(*start_routine)(void *arg);
    void *arg;
    void *status;
};

typedef struct timed_thread *timed_thread_t;
static pthread_key_t timed_thread_key;
static pthread_once_t timed_thread_once = PTHREAD_ONCE_INIT;

static void timed_thread_init()
{
    pthread_key_create(&timed_thread_key, NULL);
}

static void *timed_thread_start_routine(void *args)

/*
 * Routine to establish thread-specific data value and run the actual
 * thread start routine which was supplied to timed_thread_create().
 */
{
```



```
131336              timed_thread_t tt = (timed_thread_t) args;

131337              pthread_once(&timed_thread_once, timed_thread_init);
131338              pthread_setspecific(timed_thread_key, (void *)tt);
131339              timed_thread_exit((tt->start_routine)(tt->arg));
131340          }

131341          int timed_thread_create(timed_thread_t ttp, const pthread_attr_t *attr,
131342              void *(*start_routine)(void *), void *arg)

131343          /*
131344           * Allocate a thread which can be used with timed_thread_join().
131345           */
131346          {
131347              timed_thread_t tt;
131348              int result;

131349              tt = (timed_thread_t) malloc(sizeof(struct timed_thread));
131350              pthread_mutex_init(&tt->m,NULL);
131351              tt->exiting = FALSE;
131352              pthread_cond_init(&tt->exit_c,NULL);
131353              tt->start_routine = start_routine;
131354              tt->arg = arg;
131355              tt->status = NULL;

131356              if ((result = pthread_create(&tt->t, attr,
131357                  timed_thread_start_routine, (void *)tt)) != 0) {
131358                  free(tt);
131359                  return result;
131360              }

131361              pthread_detach(tt->t);
131362              ttp = tt;
131363              return 0;
131364          }

131365          int timed_thread_join(timed_thread_t tt,
131366              struct timespec *timeout,
131367              void **status)
131368          {
131369              int result;

131370              pthread_mutex_lock(&tt->m);
131371              result = 0;
131372              /*
131373               * Wait until the thread announces that it is exiting,
131374               * or until timeout.
131375               */
131376              while (result == 0 && ! tt->exiting) {
131377                  result = pthread_cond_timedwait(&tt->exit_c, &tt->m, timeout);
131378              }
131379              pthread_mutex_unlock(&tt->m);
131380              if (result == 0 && tt->exiting) {
131381                  *status = tt->status;
131382                  free((void *)tt);
131383                  return result;
131384              }
```



```
131385          return result;
131386      }
131387      void timed_thread_exit(void *status)
131388      {
131389          timed_thread_t tt;
131390          void *specific;
131391          if ((specific=pthread_getspecific(timed_thread_key)) == NULL){
131392              /*
131393               * Handle cases which will not happen with correct usage.
131394               */
131395              pthread_exit( NULL);
131396          }
131397          tt = (timed_thread_t) specific;
131398          pthread_mutex_lock(&tt->m);
131399          /*
131400           * Tell a joiner that we are exiting.
131401           */
131402          tt->status = status;
131403          tt->exiting = TRUE;
131404          pthread_cond_signal(&tt->exit_c);
131405          pthread_mutex_unlock(&tt->m);
131406          /*
131407           * Call pthread exit() to call destructors and really
131408           * exit the thread.
131409           */
131410          pthread_exit(NULL);
131411      }
```

131412 The *pthread_join*() C-language example shown above demonstrates that it is possible,
131413 using existing pthread facilities, to construct a variety of thread which allows for joining
131414 such a thread, but which allows the join operation to time out. It does this by using a
131415 *pthread_cond_timedwait*() to wait for the thread to exit. A **timed_thread_t** descriptor
131416 structure is used to pass parameters from the creating thread to the created thread, and
131417 from the exiting thread to the joining thread. This implementation is roughly equivalent to
131418 what a normal *pthread_join*() implementation would do, with the single change being that
131419 *pthread_cond_timedwait*() is used in place of a simple *pthread_cond_wait*().

131420 Since it is possible to implement such a facility entirely from existing pthread interfaces,
131421 and with roughly equal efficiency and complexity to an implementation which would be
131422 provided directly by a pthreads implementation, it was the consensus of the working
131423 group members that any *pthread_timedjoin*() facility would be unnecessary, and should not
131424 be provided.

131425 • Form of the Timeout Interfaces

131426 The working group considered a number of alternative ways to add timeouts to blocking
131427 services. At first, a system interface which would specify a one-shot or persistent timeout
131428 to be applied to subsequent blocking services invoked by the calling process or thread was
131429 considered because it allowed all blocking services to be timed out in a uniform manner
131430 with a single additional interface; this was rather quickly rejected because it could easily
131431 result in the wrong services being timed out.

131432 It was suggested that a timeout value might be specified as an attribute of the object
131433 (semaphore, mutex, message queue, and so on), but there was no consensus on this, either

on a case-by-case basis or for all timeouts.

Looking at the two existing timeouts for blocking services indicates that the working group members favor a separate interface for the timed version of a function. However, *pthread_cond_timedwait*() utilizes an absolute timeout value while *sigtimedwait*() uses a relative timeout value. The working group members agreed that relative timeout values are appropriate where the timeout mechanism's primary use was to deal with an unexpected or error situation, but they are inappropriate when the timeout must expire at a particular time, or before a specific deadline. For the timeouts being introduced in POSIX.1-2024, the working group considered allowing both relative and absolute timeouts as is done with POSIX.1b timers, but ultimately favored the simpler absolute timeout form.

An absolute time measure can be easily implemented on top of an interface that specifies relative time, by reading the clock, calculating the difference between the current time and the desired wakeup time, and issuing a relative timeout call. But there is a race condition with this approach because the thread could be preempted after reading the clock, but before making the timed-out call; in this case, the thread would be awakened later than it should and, thus, if the wakeup time represented a deadline, it would miss it.

There is also a race condition when trying to build a relative timeout on top of an interface that specifies absolute timeouts. In this case, the clock would have to be read to calculate the absolute wakeup time as the sum of the current time plus the relative timeout interval. In this case, if the thread is preempted after reading the clock but before making the timed-out call, the thread would be awakened earlier than desired.

But the race condition with the absolute timeouts interface is not as bad as the one that happens with the relative timeout interface, because there are simple workarounds. For the absolute timeouts interface, if the timing requirement is a deadline, the deadline can still be met because the thread woke up earlier than the deadline. If the timeout is just used as an error recovery mechanism, the precision of timing is not really important. If the timing requirement is that between actions A and B a minimum interval of time must elapse, the absolute timeout interface can be safely used by reading the clock after action A has been started. It could be argued that, since the call with the absolute timeout is atomic from the application point of view, it is not possible to read the clock after action A, if this action is part of the timed-out call. But looking at the nature of the calls for which timeouts are specified (locking a mutex, waiting for a semaphore, waiting for a message, or waiting until there is space in a message queue), the timeouts that an application would build on these actions would not be triggered by these actions themselves, but by some other external action. For example, if waiting for a message to arrive to a message queue, and waiting for at least 20 milliseconds, this time interval would start to be counted from some event that would trigger both the action that produces the message, as well as the action that waits for the message to arrive, and not by the wait-for-message operation itself. In this case, the workaround proposed above could be used.

For these reasons, the absolute timeout is preferred over the relative timeout interface.

B.2.9 Threads

Threads will normally be more expensive than subroutines (or functions, routines, and so on) if specialized hardware support is not provided. Nevertheless, threads should be sufficiently efficient to encourage their use as a medium to fine-grained structuring mechanism for parallelism in an application. Structuring an application using threads then allows it to take immediate advantage of any underlying parallelism available in the host environment. This means implementors are encouraged to optimize for fast execution at the possible expense of efficient utilization of storage. For example, a common thread creation technique is to cache appropriate thread data structures. That is, rather than releasing system resources, the implementation retains these resources and reuses them when the program next asks to create a new thread. If this reuse of thread resources is to be possible, there has to be very little unique state associated with each thread, because any such state has to be reset when the thread is reused.

Thread Creation Attributes

Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to support probable future standardization in these areas without requiring that the interface itself be changed.

Attributes objects provide clean isolation of the configurable aspects of threads. For example, ``stack size'' is an important attribute of a thread, but it cannot be expressed portably. When porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects can help by allowing the changes to be isolated in a single place, rather than being spread across every instance of thread creation.

Attributes objects can be used to set up *classes* of threads with similar attributes; for example, ``threads with large stacks and high priority'' or ``threads with minimal stacks''. These classes can be defined in a single place and then referenced wherever threads need to be created. Changes to ``class'' decisions become straightforward, and detailed analysis of each *pthread_create*( ) call is not required.

The attributes objects are defined as opaque types as an aid to extensibility. If these objects had been specified as structures, adding new attributes would force recompilation of all multi-threaded programs when the attributes objects are extended; this might not be possible if different program components were supplied by different vendors.

Additionally, opaque attributes objects present opportunities for improving performance. Argument validity can be checked once when attributes are set, rather than each time a thread is created. Implementations will often need to cache kernel objects that are expensive to create. Opaque attributes objects provide an efficient mechanism to detect when cached objects become invalid due to attribute changes.

Because assignment is not necessarily defined on a given opaque type, implementation-defined default values cannot be defined in a portable way. The solution to this problem is to allow attribute objects to be initialized dynamically by attributes object initialization functions, so that default values can be supplied automatically by the implementation.

The following proposal was provided as a suggested alternative to the supplied attributes:

1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to the initialization routines (*pthread_create*( ), *pthread_mutex_init*( ), *pthread_cond_init*( )). The parameter containing the flags should be an opaque type for extensibility. If no flags are set in the parameter, then the objects are created with default characteristics. An implementation may specify implementation-defined flag values and associated behavior.

2. If further specialization of mutexes and condition variables is necessary, implementations may specify additional procedures that operate on the **pthread_mutex_t** and **pthread_cond_t** objects (instead of on attributes objects).

The difficulties with this solution are:

1. A bitmask is not opaque if bits have to be set into bit-vector attributes objects using explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**, application programmers need to know the location of each bit. If bits are set or read by encapsulation (that is, *get**() or *set**() functions), then the bitmask is merely an implementation of attributes objects as currently defined and should not be exposed to the programmer.

2. Many attributes are not Boolean or very small integral values. For example, scheduling policy may be placed in 3 bits or 4 bits, but priority requires 5 bits or more, thereby taking up at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this, the bitmask can only reasonably control whether particular attributes are set or not, and it cannot serve as the repository of the value itself. The value needs to be specified as a function parameter (which is non-extensible), or by setting a structure field (which is non-opaque), or by *get**() and *set**() functions (making the bitmask a redundant addition to the attributes objects).

Stack size is defined as an optional attribute because the very notion of a stack is inherently machine-dependent. Some implementations may not be able to change the size of the stack, for example, and others may not need to because stack pages may be discontiguous and can be allocated and released on demand.

The attribute mechanism has been designed in large measure for extensibility. Future extensions to the attribute mechanism or to any attributes object defined in POSIX.1-2024 have to be done with care so as not to affect binary-compatibility.

Attribute objects, even if allocated by means of dynamic allocation functions such as *malloc*(), may have their size fixed at compile time. This means, for example, a *pthread_create*() in an implementation with extensions to the **pthread_attr_t** cannot look beyond the area that the binary application assumes is valid. This suggests that implementations should maintain a size field in the attributes object, as well as possibly version information, if extensions in different directions (possibly by different vendors) are to be accommodated.

Thread Implementation Models

There are various thread implementation models. At one end of the spectrum is the ``library-thread model''. In such a model, the threads of a process are not visible to the operating system kernel, and the threads are not kernel-scheduled entities. The process is the only kernel-scheduled entity. The process is scheduled onto the processor by the kernel according to the scheduling attributes of the process. The threads are scheduled onto the single kernel-scheduled entity (the process) by the runtime library according to the scheduling attributes of the threads. A problem with this model is that it constrains concurrency. Since there is only one kernel-scheduled entity (namely, the process), only one thread per process can execute at a time. If the thread that is executing blocks on I/O, then the whole process blocks.

At the other end of the spectrum is the ``kernel-thread model''. In this model, all threads are visible to the operating system kernel. Thus, all threads are kernel-scheduled entities, and all threads can concurrently execute. The threads are scheduled onto processors by the kernel according to the scheduling attributes of the threads. The drawback to this model is that the creation and management of the threads entails operating system calls, as opposed to subroutine calls, which makes kernel threads heavier weight than library threads.

Hybrids of these two models are common. A hybrid model offers the speed of library threads and the concurrency of kernel threads. In hybrid models, a process has some (relatively small) number of kernel scheduled entities associated with it. It also has a potentially much larger number of library threads associated with it. Some library threads may be bound to kernel-scheduled entities, while the other library threads are multiplexed onto the remaining kernel-scheduled entities. There are two levels of thread scheduling:

1. The runtime library manages the scheduling of (unbound) library threads onto kernel-scheduled entities.
2. The kernel manages the scheduling of kernel-scheduled entities onto processors.

For this reason, a hybrid model is referred to as a two-level threads scheduling model. In this model, the process can have multiple concurrently executing threads; specifically, it can have as many concurrently executing threads as it has kernel-scheduled entities.

Thread-Specific Data

Many applications require that a certain amount of context be maintained on a per-thread basis across procedure calls. A common example is a multi-threaded library routine that allocates resources from a common pool and maintains an active resource list for each thread. The thread-specific data interface provided to meet these needs may be viewed as a two-dimensional array of values with keys serving as the row index and thread IDs as the column index (although the implementation need not work this way).

- Models

  Three possible thread-specific data models were considered:

  1. No Explicit Support

     A standard thread-specific data interface is not strictly necessary to support applications that require per-thread context. One could, for example, provide a hash function that converted a **pthread_t** into an integer value that could then be used to index into a global array of per-thread data pointers. This hash function, in conjunction with *pthread_self*( ), would be all the interface required to support a mechanism of this sort. Unfortunately, this technique is cumbersome. It can lead to duplicated code as each set of cooperating modules implements their own per-thread data management schemes. This technique would also require that **pthread_t** not be an opaque type.

  2. Single (**void** *) Pointer

     Another technique would be to provide a single word of per-thread storage and a pair of functions to fetch and store the value of this word. The word could then hold a pointer to a block of per-thread memory. The allocation, partitioning, and general use of this memory would be entirely up to the application. Although this method is not as problematic as technique 1, it suffers from interoperability problems. For example, all modules using the per-thread pointer would have to agree on a common usage protocol.

  3. Key/Value Mechanism

     This method associates an opaque key (for example, stored in a variable of type **pthread_key_t**) with each per-thread datum. These keys play the role of identifiers for per-thread data. This technique is the most generic and avoids the problems noted above, albeit at the cost of some complexity.

  The primary advantage of the third model is its information hiding properties. Modules

using this model are free to create and use their own key(s) independent of all other such usage, whereas the other models require that all modules that use thread-specific context explicitly cooperate with all other such modules. The data-independence provided by the third model is worth the additional interface. Therefore, the third model was chosen.

- Requirements

It is important that it be possible to implement the thread-specific data interface without the use of thread private memory. To do otherwise would increase the weight of each thread, thereby limiting the range of applications for which the threads interfaces provided by POSIX.1-2024 is appropriate.

The values that one binds to the key via *pthread_setspecific*() may, in fact, be pointers to shared storage locations available to all threads. It is only the key/value bindings that are maintained on a per-thread basis, and these can be kept in any portion of the address space that is reserved for use by the calling thread (for example, on the stack). Thus, no per-thread MMU state is required to implement the interface. On the other hand, there is nothing in the interface specification to preclude the use of a per-thread MMU state if it is available (for example, the key values returned by *pthread_key_create*() could be thread private memory addresses).

- Standardization Issues

Thread-specific data is a requirement for a usable thread interface. The binding described in this section provides a portable thread-specific data mechanism for languages that do not directly support a thread-specific storage class. A binding to POSIX.1-2024 for a language that does include such a storage class need not provide this specific interface.

If a language were to include the notion of thread-specific storage, it would be desirable (but *not* required) to provide an implementation of the pthreads thread-specific data interface based on the language feature. For example, assume that a compiler for a C-like language supports a *private* storage class that provides thread-specific storage. Something similar to the following macros might be used to effect a compatible implementation:

```
#define pthread_key_t                   private void *
#define pthread_key_create(key)         /* no-op */
#define pthread_setspecific(key,value)  (key)=(value)
#define pthread_getspecific(key)        (key)
```

Note: For the sake of clarity, this example ignores destructor functions. A correct implementation would have to support them.

Barriers

- Background

Barriers are typically used in parallel DO/FOR loops to ensure that all threads have reached a particular stage in a parallel computation before allowing any to proceed to the next stage. Highly efficient implementation is possible on machines which support a ``Fetch and Add'' operation as described in the referenced Almasi and Gottlieb (1989).

The use of return value PTHREAD_BARRIER_SERIAL_THREAD is shown in the following example:

```
if ( (status=pthread_barrier_wait(&barrier)) ==
    PTHREAD_BARRIER_SERIAL_THREAD) {
    ...serial section
}
else if (status != 0) {
```



```
        ...error processing
    }
    status=pthread_barrier_wait(&barrier);
    ...
```

This behavior allows a serial section of code to be executed by one thread as soon as all threads reach the first barrier. The second barrier prevents the other threads from proceeding until the serial section being executed by the one thread has completed.

Although barriers can be implemented with mutexes and condition variables, the referenced Almasi and Gottlieb (1989) provides ample illustration that such implementations are significantly less efficient than is possible. While the relative efficiency of barriers may well vary by implementation, it is important that they be recognized in the POSIX.1-2024 to facilitate applications portability while providing the necessary freedom to implementors.

- Lack of Timeout Feature

Alternate versions of most blocking routines have been provided to support watchdog timeouts. No alternate interface of this sort has been provided for barrier waits for the following reasons:

  - Multiple threads may use different timeout values, some of which may be indefinite. It is not clear which threads should break through the barrier with a timeout error if and when these timeouts expire.
  - The barrier may become unusable once a thread breaks out of a *pthread_barrier_wait*() with a timeout error. There is, in general, no way to guarantee the consistency of a barrier's internal data structures once a thread has timed out of a *pthread_barrier_wait*(). Even the inclusion of a special barrier reinitialization function would not help much since it is not clear how this function would affect the behavior of threads that reach the barrier between the original timeout and the call to the reinitialization function.

Spin Locks

- Background

Spin locks represent an extremely low-level synchronization mechanism suitable primarily for use on shared memory multi-processors. It is typically an atomically modified Boolean value that is set to one when the lock is held and to zero when the lock is freed.

When a caller requests a spin lock that is already held, it typically spins in a loop testing whether the lock has become available. Such spinning wastes processor cycles so the lock should only be held for short durations and not across sleep/block operations. Callers should unlock spin locks before calling sleep operations.

Spin locks are available on a variety of systems. The functions included in POSIX.1-2024 are an attempt to standardize that existing practice.

- Lack of Timeout Feature

Alternate versions of most blocking routines have been provided to support watchdog timeouts. No alternate interface of this sort has been provided for spin locks for the following reasons:

  - It is impossible to determine appropriate timeout intervals for spin locks in a portable manner. The amount of time one can expect to spend spin-waiting is inversely proportional to the degree of parallelism provided by the system.

131704 It can vary from a few cycles when each competing thread is running on its own
131705 processor, to an indefinite amount of time when all threads are multiplexed on a
131706 single processor (which is why spin locking is not advisable on uniprocessors).

131707 • When used properly, the amount of time the calling thread spends waiting on a spin
131708 lock should be considerably less than the time required to set up a corresponding
131709 watchdog timer. Since the primary purpose of spin locks is to provide a low-
131710 overhead synchronization mechanism for multi-processors, the overhead of a
131711 timeout mechanism was deemed unacceptable.

131712 It was also suggested that an additional *count* argument be provided (on the
131713 *pthread_spin_lock*( ) call) in *lieu* of a true timeout so that a spin lock call could fail gracefully
131714 if it was unable to apply the lock after *count* attempts. This idea was rejected because it is
131715 not existing practice. Furthermore, the same effect can be obtained with
131716 *pthread_spin_trylock*( ), as illustrated below:

131717
```
int n = MAX_SPIN;
```

131718
```
while ( --n >= 0 )
{
    if ( !pthread_spin_try_lock(...) )
        break;
}
if ( n >= 0 )
{
    /* Successfully acquired the lock */
}
else
{
    /* Unable to acquire the lock */
}
```

131731 • *process-shared* Attribute

131732 The initialization functions associated with most POSIX synchronization objects (for
131733 example, mutexes, barriers, and read-write locks) take an attributes object with a *process-*
131734 *shared* attribute that specifies whether or not the object is to be shared across processes. In
131735 the draft corresponding to the first balloting round, two separate initialization functions
131736 are provided for spin locks, however: one for spin locks that were to be shared across
131737 processes (*spin_init*( )), and one for locks that were only used by multiple threads within a
131738 single process (*pthread_spin_init*( )). This was done so as to keep the overhead associated
131739 with spin waiting to an absolute minimum. However, the balloting group requested that,
131740 since the overhead associated to a bit check was small, spin locks should be consistent with
131741 the rest of the synchronization primitives, and thus the *process-shared* attribute was
131742 introduced for spin locks.

131743 • Spin Locks *versus* Mutexes

131744 It has been suggested that mutexes are an adequate synchronization mechanism and spin
131745 locks are not necessary. Locking mechanisms typically must trade off the processor
131746 resources consumed while setting up to block the thread and the processor resources
131747 consumed by the thread while it is blocked. Spin locks require very little resources to set
131748 up the blocking of a thread. Existing practice is to simply loop, repeating the atomic
131749 locking operation until the lock is available. While the resources consumed to set up
131750 blocking of the thread are low, the thread continues to consume processor resources while
131751 it is waiting.

On the other hand, mutexes may be implemented such that the processor resources consumed to block the thread are large relative to a spin lock. After detecting that the mutex lock is not available, the thread must alter its scheduling state, add itself to a set of waiting threads, and, when the lock becomes available again, undo all of this before taking over ownership of the mutex. However, while a thread is blocked by a mutex, no processor resources are consumed.

Therefore, spin locks and mutexes may be implemented to have different characteristics. Spin locks may have lower overall overhead for very short-term blocking, and mutexes may have lower overall overhead when a thread will be blocked for longer periods of time. The presence of both interfaces allows implementations with these two different characteristics, both of which may be useful to a particular application.

It has also been suggested that applications can build their own spin locks from the *pthread_mutex_trylock*( ) function:

```
while (pthread_mutex_trylock(&mutex));
```

The apparent simplicity of this construct is somewhat deceiving, however. While the actual wait is quite efficient, various guarantees on the integrity of mutex objects (for example, priority inheritance rules) may add overhead to the successful path of the trylock operation that is not required of spin locks. One could, of course, add an attribute to the mutex to bypass such overhead, but the very act of finding and testing this attribute represents more overhead than is found in the typical spin lock.

The need to hold spin lock overhead to an absolute minimum also makes it impossible to provide guarantees against starvation similar to those provided for mutexes or read-write locks. The overhead required to implement such guarantees (for example, disabling preemption before spinning) may well exceed the overhead of the spin wait itself by many orders of magnitude. If a ``safe'' spin wait seems desirable, it can always be provided (albeit at some performance cost) via appropriate mutex attributes.

Robust Mutexes

Robust mutexes are intended to protect applications that use mutexes to protect data shared between different processes. If a process is terminated by a signal while a thread is holding a mutex, there is no chance for the process to clean up after it. Waiters for the locked mutex might wait indefinitely.

With robust mutexes the problem can be solved: whenever a fatal signal terminates a process, current or future waiters of the mutex are notified about this fact. The locking function provides notification of this condition through the error condition [EOWNERDEAD]. A thread then has the chance to clean up the state protected by the mutex and mark the state as consistent again by a call to *pthread_mutex_consistent*( ).

Pre-existing implementations have used the semantics of robust mutexes for a variety of situations, some of them not defined in the standard. Where a normally terminated process (i.e., when one thread calls *exit*( )) causes notification of other waiters of robust mutexes if the mutex is locked by any thread in the process. This behavior is defined in the standard and makes sense because no thread other than the thread calling *exit*( ) has the chance to clean up its data.

If a thread is terminated by cancellation or if it calls *pthread_exit*( ), the situation is different. In both these situations the thread has the chance to clean up after itself by registering appropriate cleanup handlers. There is no real reason to demand that other waiters for a robust mutex the terminating thread owns are notified. The committee felt that this is actively encouraging bad practice because programmers are tempted to rely on the robust mutex semantics instead of correctly cleaning up after themselves.

Therefore, the standard does not require notification of other waiters at the time a thread is terminated while the process continues to run. The mutex is still recognized as being locked by the process (with the thread gone it makes no sense to refer to the thread owning the mutex). Therefore, a terminating process will cause notifications about the dead owner to be sent to all waiters. This delay in the notification is not required, but programmers cannot rely on prompt notification after a thread is terminated.

For the same reason is it not required that an implementation supports robust mutexes that are not shared between processes. If a robust mutex is used only within one process, all the cleanup can be performed by the threads themselves by registering appropriate cleanup handlers. Fatal signals are of no importance in this case because after the signal is delivered there is no thread remaining to use the mutex.

Some implementations might choose to support intra-process robust mutexes and they might also send notification of a dead owner right after the previous owner died. But applications must not rely on this. Applications should only use robust mutexes for the purpose of handling fatal signals in situations where inter-process mutexes are in use.

Supported Threads Functions

On POSIX-conforming systems, the following symbolic constants are always conforming:

_POSIX_READER_WRITER_LOCKS
_POSIX_THREADS

Therefore, the following threads functions are always supported:

pthread_atfork()
pthread_attr_destroy()
pthread_attr_getdetachstate()
pthread_attr_getguardsize()
pthread_attr_getschedparam()
pthread_attr_init()
pthread_attr_setdetachstate()
pthread_attr_setguardsize()
pthread_attr_setschedparam()
pthread_cancel()
pthread_cleanup_pop()
pthread_cleanup_push()
pthread_cond_broadcast()
pthread_cond_clockwait()
pthread_cond_destroy()
pthread_cond_init()
pthread_cond_signal()
pthread_cond_timedwait()
pthread_cond_wait()
pthread_condattr_destroy()
pthread_condattr_getpshared()
pthread_condattr_init()
pthread_condattr_setpshared()
pthread_create()
pthread_detach()
pthread_equal()
pthread_exit()
pthread_getspecific()
pthread_join()
pthread_key_create()
pthread_key_delete()
pthread_kill()
pthread_mutex_destroy()
pthread_mutex_init()
pthread_mutex_lock()
pthread_mutex_trylock()
pthread_mutex_unlock()
pthread_mutexattr_destroy()
pthread_mutexattr_getpshared()
pthread_mutexattr_gettype()
pthread_mutexattr_init()
pthread_mutexattr_setpshared()
pthread_mutexattr_settype()
pthread_once()
pthread_rwlock_destroy()
pthread_rwlock_init()
pthread_rwlock_rdlock()
pthread_rwlock_tryrdlock()
pthread_rwlock_trywrlock()
pthread_rwlock_unlock()
pthread_rwlock_wrlock()
pthread_rwlockattr_destroy()
pthread_rwlockattr_getpshared()
pthread_rwlockattr_init()
pthread_rwlockattr_setpshared()
pthread_self()
pthread_setcancelstate()
pthread_setcanceltype()
pthread_setspecific()
pthread_sigmask()
pthread_testcancel()
sigwait()

131850 On POSIX-conforming systems, the symbolic constant _POSIX_THREAD_SAFE_FUNCTIONS is
131851 always defined. Therefore, the following functions are always supported:

131852 *flockfile*( )	*getpwuid_r*( )
131853 *ftrylockfile*( )	*gmtime_r*( )
131854 *funlockfile*( )	*localtime_r*( )
131855 *getc_unlocked*( )	*putc_unlocked*( )
131856 *getchar_unlocked*( )	*putchar_unlocked*( )
131857 *getgrgid_r*( )	*readdir_r*( )
131858 *getgrnam_r*( )	*strerror_r*( )
131859 *getpwnam_r*( )	*strtok_r*( )

131860 **Threads Extensions**

131861 The following extensions to the IEEE P1003.1c draft standard are now supported in
131862 POSIX.1-2024 as part of the alignment with the Single UNIX Specification:

131863 • Extended mutex attribute types

131864 • Read-write locks and attributes (also introduced by the IEEE Std 1003.1j-2000 amendment)

131865 • Thread concurrency level

131866 • Thread stack guard size

131867 • Parallel I/O

131868 • Robust mutexes

131869 These extensions carefully follow the threads programming model specified in POSIX.1c. As
131870 with POSIX.1c, all the new functions return zero if successful; otherwise, an error number is
131871 returned to indicate the error.

131872 The concept of attribute objects was introduced in POSIX.1c to allow implementations to extend
131873 POSIX.1-2024 without changing the existing interfaces. Attribute objects were defined for
131874 threads, mutexes, and condition variables. Attributes objects are defined as implementation-
131875 defined opaque types to aid extensibility, and functions are defined to allow attributes to be set
131876 or retrieved. This model has been followed when adding the new type attribute of
131877 **pthread_mutexattr_t** or the new read-write lock attributes object **pthread_rwlockattr_t**.

131878 • Extended Mutex Attributes

131879 POSIX.1c defines a mutex attributes object as an implementation-defined opaque object of
131880 type **pthread_mutexattr_t**, and specifies a number of attributes which this object must
131881 have and a number of functions which manipulate these attributes. These attributes
131882 include *detachstate*, *inheritsched*, *schedparm*, *schedpolicy*, *contentionscope*, *stackaddr*, and
131883 *stacksize*.

131884 The System Interfaces volume of POSIX.1-2024 specifies another mutex attribute called
131885 *type*. The *type* attribute allows applications to specify the behavior of mutex locking
131886 operations in situations where POSIX.1c behavior is undefined. The OSF DCE threads
131887 implementation, based on Draft 4 of POSIX.1c, specified a similar attribute. Note that the
131888 names of the attributes have changed somewhat from the OSF DCE threads
131889 implementation.

131890 The System Interfaces volume of POSIX.1-2024 also extends the specification of the
131891 following POSIX.1c functions which manipulate mutexes:

pthread_mutex_lock( )
pthread_mutex_trylock( )
pthread_mutex_unlock( )

to take account of the new mutex attribute type and to specify behavior which was declared as undefined in POSIX.1c. How a calling thread acquires or releases a mutex now depends upon the mutex *type* attribute.

The *type* attribute can have the following values:

PTHREAD_MUTEX_NORMAL
Basic mutex with no specific error checking built in. Does not report a deadlock error.

PTHREAD_MUTEX_RECURSIVE
Allows any thread to recursively lock a mutex. The mutex must be unlocked an equal number of times to release the mutex.

PTHREAD_MUTEX_ERRORCHECK
Detects and reports simple usage errors; that is, an attempt to unlock a mutex that is not locked by the calling thread or that is not locked at all, or an attempt to relock a mutex the thread already owns.

PTHREAD_MUTEX_DEFAULT
The default mutex type. May be mapped to any of the above mutex types or may be an implementation-defined type.

Normal mutexes do not detect deadlock conditions; for example, a thread will hang if it tries to relock a normal mutex that it already owns. Attempting to unlock a mutex locked by another thread, or unlocking an unlocked mutex, results in undefined behavior. Normal mutexes will usually be the fastest type of mutex available on a platform but provide the least error checking.

Recursive mutexes are useful for converting old code where it is difficult to establish clear boundaries of synchronization. A thread can relock a recursive mutex without first unlocking it. The relocking deadlock which can occur with normal mutexes cannot occur with this type of mutex. However, multiple locks of a recursive mutex require the same number of unlocks to release the mutex before another thread can acquire the mutex. Furthermore, this type of mutex maintains the concept of an owner. Thus, a thread attempting to unlock a recursive mutex which another thread has locked returns with an error. A thread attempting to unlock a recursive mutex that is not locked returns with an error. Never use a recursive mutex with condition variables because the implicit unlock performed by *pthread_cond_clockwait*( ), *pthread_cond_timedwait*( ), or *pthread_cond_wait*( ) will not actually release the mutex if it had been locked multiple times.

Errorcheck mutexes provide error checking and are useful primarily as a debugging aid. A thread attempting to relock an errorcheck mutex without first unlocking it returns with an error. Again, this type of mutex maintains the concept of an owner. Thus, a thread attempting to unlock an errorcheck mutex which another thread has locked returns with an error. A thread attempting to unlock an errorcheck mutex that is not locked also returns with an error. It should be noted that errorcheck mutexes will almost always be much slower than normal mutexes due to the extra state checks performed.

The default mutex type provides implementation-defined error checking. The default mutex may be mapped to one of the other defined types or may be something entirely different. This enables each vendor to provide the mutex semantics which the vendor feels will be most useful to their target users. Most vendors will probably choose to make normal mutexes the default so as to give applications the benefit of the fastest type of

131939 mutexes available on their platform. Check your implementation's documentation.

131940 An application developer can use any of the mutex types almost interchangeably as long
131941 as the application does not depend upon the implementation detecting (or failing to
131942 detect) any particular errors. Note that a recursive mutex can be used with condition
131943 variable waits as long as the application never recursively locks the mutex.

131944 Two functions are provided for manipulating the *type* attribute of a mutex attributes object.
131945 This attribute is set or returned in the *type* parameter of these functions. The
131946 *pthread_mutexattr_settype*( ) function is used to set a specific type value while
131947 *pthread_mutexattr_gettype*( ) is used to return the type of the mutex. Setting the *type*
131948 attribute of a mutex attributes object affects only mutexes initialized using that mutex
131949 attributes object. Changing the *type* attribute does not affect mutexes previously initialized
131950 using that mutex attributes object.

131951 • Read-Write Locks and Attributes

131952 The read-write locks introduced have been harmonized with those in IEEE Std
131953 1003.1j-2000; see also Section B.2.9.6 (on page 3833).

131954 Read-write locks (also known as reader-writer locks) allow a thread to exclusively lock
131955 some shared data while updating that data, or allow any number of threads to have
131956 simultaneous read-only access to the data.

131957 Unlike a mutex, a read-write lock distinguishes between reading data and writing data. A
131958 mutex excludes all other threads. A read-write lock allows other threads access to the data,
131959 providing no thread is modifying the data. Thus, a read-write lock is less primitive than
131960 either a mutex-condition variable pair or a semaphore.

131961 Application developers should consider using a read-write lock rather than a mutex to
131962 protect data that is frequently referenced but seldom modified. Most threads (readers) will
131963 be able to read the data without waiting and will only have to block when some other
131964 thread (a writer) is in the process of modifying the data. Conversely a thread that wants to
131965 change the data is forced to wait until there are no readers. This type of lock is often used
131966 to facilitate parallel access to data on multi-processor platforms or to avoid context
131967 switches on single processor platforms where multiple threads access the same data.

131968 If a read-write lock becomes unlocked and there are multiple threads waiting to acquire
131969 the write lock, the implementation's scheduling policy determines which thread acquires
131970 the read-write lock for writing. If there are multiple threads blocked on a read-write lock
131971 for both read locks and write locks, it is unspecified whether the readers or a writer
131972 acquire the lock first. However, for performance reasons, implementations often favor
131973 writers over readers to avoid potential writer starvation.

131974 A read-write lock object is an implementation-defined opaque object of type
131975 **pthread_rwlock_t** as defined in **<pthread.h>**. There are two different sorts of locks
131976 associated with a read-write lock: a read lock and a write lock.

131977 The *pthread_rwlockattr_init*( ) function initializes a read-write lock attributes object with the
131978 default value for all the attributes defined in the implementation. After a read-write lock
131979 attributes object has been used to initialize one or more read-write locks, changes to the
131980 read-write lock attributes object, including destruction, do not affect previously initialized
131981 read-write locks.

131982 Implementations must provide at least the read-write lock attribute *process-shared*. This
131983 attribute can have the following values:

PTHREAD_PROCESS_SHARED
Any thread of any process that has access to the memory where the read-write lock resides can manipulate the read-write lock.

PTHREAD_PROCESS_PRIVATE
Only threads created within the same process as the thread that initialized the read-write lock can manipulate the read-write lock. This is the default value.

The *pthread_rwlockattr_setpshared*( ) function is used to set the *process-shared* attribute of an initialized read-write lock attributes object while the function *pthread_rwlockattr_getpshared*( ) obtains the current value of the *process-shared* attribute.

A read-write lock attributes object is destroyed using the *pthread_rwlockattr_destroy*( ) function. The effect of subsequent use of the read-write lock attributes object is undefined.

A thread creates a read-write lock using the *pthread_rwlock_init*( ) function. The attributes of the read-write lock can be specified by the application developer; otherwise, the default implementation-defined read-write lock attributes are used if the pointer to the read-write lock attributes object is NULL. In cases where the default attributes are appropriate, the PTHREAD_RWLOCK_INITIALIZER macro can be used to initialize read-write locks.

A thread which wants to apply a read lock to the read-write lock can use either *pthread_rwlock_rdlock*( ) or *pthread_rwlock_tryrdlock*( ). If *pthread_rwlock_rdlock*( ) is used, the thread acquires a read lock if a writer does not hold the write lock and there are no writers blocked on the write lock. If a read lock is not acquired, the calling thread blocks until it can acquire a lock. However, if *pthread_rwlock_tryrdlock*( ) is used, the function returns immediately with the error [EBUSY] if any thread holds a write lock or there are blocked writers waiting for the write lock.

A thread which wants to apply a write lock to the read-write lock can use either of two functions: *pthread_rwlock_wrlock*( ) or *pthread_rwlock_trywrlock*( ). If *pthread_rwlock_wrlock*( ) is used, the thread acquires the write lock if no other reader or writer threads hold the read-write lock. If the write lock is not acquired, the thread blocks until it can acquire the write lock. However, if *pthread_rwlock_trywrlock*( ) is used, the function returns immediately with the error [EBUSY] if any thread is holding either a read or a write lock.

The *pthread_rwlock_unlock*( ) function is used to unlock a read-write lock object held by the calling thread. Results are undefined if the read-write lock is not held by the calling thread. If there are other read locks currently held on the read-write lock object, the read-write lock object remains in the read locked state but without the current thread as one of its owners. If this function releases the last read lock for this read-write lock object, the read-write lock object is put in the unlocked read state. If this function is called to release a write lock for this read-write lock object, the read-write lock object is put in the unlocked state.

- Thread Concurrency Level

On threads implementations that multiplex user threads onto a smaller set of kernel execution entities, the system attempts to create a reasonable number of kernel execution entities for the application upon application startup.

On some implementations, these kernel entities are retained by user threads that block in the kernel. Other implementations do not *timeslice* user threads so that multiple compute-bound user threads can share a kernel thread. On such implementations, some applications may use up all the available kernel execution entities before their user-space threads are used up. The process may be left with user threads capable of doing work for the application but with no way to schedule them.

- Thread Stack Guard Size

  DCE threads introduced the concept of a ``thread stack guard size''. Most thread implementations add a region of protected memory to a thread's stack, commonly known as a ``guard region'', as a safety measure to prevent stack pointer overflow in one thread from corrupting the contents of another thread's stack. The default size of the guard regions attribute is {PAGESIZE} bytes and is implementation-defined.

  Some application developers may wish to change the stack guard size. When an application creates a large number of threads, the extra page allocated for each stack may strain system resources. In addition to the extra page of memory, the kernel's memory manager has to keep track of the different protections on adjoining pages. When this is a problem, the application developer may request a guard size of 0 bytes to conserve system resources by eliminating stack overflow protection.

  Conversely an application that allocates large data structures such as arrays on the stack may wish to increase the default guard size in order to detect stack overflow. If a thread allocates two pages for a data array, a single guard page provides little protection against thread stack overflows since the thread can corrupt adjoining memory beyond the guard page.

  The System Interfaces volume of POSIX.1-2024 defines a new attribute of a thread attributes object; that is, the *guardsize* attribute which allows applications to specify the size of the guard region of a thread's stack.

  Two functions are provided for manipulating a thread's stack guard size. The *pthread_attr_setguardsize*( ) function sets the thread *guardsize* attribute, and the *pthread_attr_getguardsize*( ) function retrieves the current value.

  An implementation may round up the requested guard size to a multiple of the configurable system variable {PAGESIZE}. In this case, *pthread_attr_getguardsize*( ) returns the guard size specified by the previous *pthread_attr_setguardsize*( ) function call and not the rounded up value.

  If an application is managing its own thread stacks using the *stackaddr* attribute, the *guardsize* attribute is ignored and no stack overflow protection is provided. In this case, it is the responsibility of the application to manage stack overflow along with stack allocation.

- Parallel I/O

  Suppose two or more threads independently issue read requests on the same file. To read specific data from a file, a thread must first call *lseek*( ) to seek to the proper offset in the file, and then call *read*( ) to retrieve the required data. If more than one thread does this at the same time, the first thread may complete its seek call, but before it gets a chance to issue its read call a second thread may complete its seek call, resulting in the first thread accessing incorrect data when it issues its read call. One workaround is to lock the file descriptor while seeking and reading or writing, but this reduces parallelism and adds overhead.

  Instead, the System Interfaces volume of POSIX.1-2024 provides two functions to make seek/read and seek/write operations atomic. The file descriptor's current offset is unchanged, thus allowing multiple read and write operations to proceed in parallel. This improves the I/O performance of threaded applications. The *pread*( ) function is used to do an atomic read of data from a file into a buffer. Conversely, the *pwrite*( ) function does an atomic write of data from a buffer to a file.

B.2.9.1 Thread-Safety

All functions required by POSIX.1-2024 need to be thread-safe. Implementations have to provide internal synchronization when necessary in order to achieve this goal. In certain cases—for example, most floating-point implementations—context switch code may have to manage the writable shared state.

While a read from a pipe of {PIPE_BUF}*2 bytes may not generate a single atomic and thread-safe stream of bytes, it should generate ``several'' (individually atomic) thread-safe streams of bytes. Similarly, while reading from a terminal device may not generate a single atomic and thread-safe stream of bytes, it should generate some finite number of (individually atomic) and thread-safe streams of bytes. That is, concurrent calls to read for a pipe, FIFO, or terminal device are not allowed to result in corrupting the stream of bytes or other internal data. However, *read*( ), in these cases, is not required to return a single contiguous and atomic stream of bytes.

It is not required that all functions provided by POSIX.1-2024 be either async-cancel-safe or async-signal-safe.

As it turns out, some functions are inherently not thread-safe; that is, their interface specifications preclude thread-safety. For example, some functions (such as *asctime*( )) return a pointer to a result stored in memory space allocated by the function on a per-process basis. Such a function is not thread-safe, because its result can be overwritten by successive invocations. Other functions, while not inherently non-thread-safe, may be implemented in ways that lead to them not being thread-safe. For example, some functions (such as *rand*( )) store state information (such as a seed value, which survives multiple function invocations) in memory space allocated by the function on a per-process basis. The implementation of such a function is not thread-safe if the implementation fails to synchronize invocations of the function and thus fails to protect the state information. The problem is that when the state information is not protected, concurrent invocations can interfere with one another (for example, applications using *rand*( ) may see the same seed value).

Thread-Safety and Locking of Existing Functions

Originally, POSIX.1 was not designed to work in a multi-threaded environment, and some implementations of some existing functions will not work properly when executed concurrently. To provide routines that will work correctly in an environment with threads (``thread-safe''), two problems need to be solved:

1. Routines that maintain or return pointers to static areas internal to the routine (which may now be shared) need to be modified. The routines *ttyname*( ) and *localtime*( ) are examples.

2. Routines that access data space shared by more than one thread need to be modified. The *malloc*( ) function and the *stdio* family routines are examples.

There are a variety of constraints on these changes. The first is compatibility with the existing versions of these functions—non-thread-safe functions will continue to be in use for some time, as the original interfaces are used by existing code. Another is that the new thread-safe versions of these functions represent as small a change as possible over the familiar interfaces provided by the existing non-thread-safe versions. The new interfaces should be independent of any particular threads implementation. In particular, they should be thread-safe without depending on explicit thread-specific memory. Finally, there should be minimal performance penalty due to the changes made to the functions.

It is intended that the list of functions from POSIX.1 that cannot be made thread-safe and for which corrected versions are provided be complete.

Thread-Safety and Locking Solutions

Many of the POSIX.1 functions were thread-safe and did not change at all. However, some functions (for example, the math functions typically found in **libm**) are not thread-safe because of writable shared global state. For instance, in IEEE Std 754-1985 floating-point implementations, the computation modes and flags are global and shared.

Some functions are not thread-safe because a particular implementation is not reentrant, typically because of a non-essential use of static storage. These require only a new implementation.

Thread-safe libraries are useful in a wide range of parallel (and asynchronous) programming environments, not just within pthreads. In order to be used outside the context of pthreads, however, such libraries still have to use some synchronization method. These could either be independent of the pthread synchronization operations, or they could be a subset of the pthread interfaces. Either method results in thread-safe library implementations that can be used without the rest of pthreads.

Some functions, such as the *stdio* family interface and dynamic memory allocation functions such as *malloc*(), are inter-dependent routines that share resources (for example, buffers) across related calls. These require synchronization to work correctly, but they do not require any change to their external (user-visible) interfaces.

In some cases, such as *getc*() and *putc*(), adding synchronization is likely to create an unacceptable performance impact. In this case, slower thread-safe synchronized functions are to be provided, but the original, faster (but unsafe) functions (which may be implemented as macros) are retained under new names. Some additional special-purpose synchronization facilities are necessary for these macros to be usable in multi-threaded programs. This also requires changes in **<stdio.h>**.

The other common reason that functions are unsafe is that they return a pointer to static storage, making the functions non-thread-safe. This has to be changed, and there are three natural choices:

1. Return a pointer to thread-specific storage

   This could incur a severe performance penalty on those architectures with a costly implementation of the thread-specific data interface.

   A variation on this technique is to use *malloc*() to allocate storage for the function output and return a pointer to this storage. This technique may also have an undesirable performance impact, however, and a simplistic implementation requires that the user program explicitly free the storage object when it is no longer needed. This technique is used by some existing POSIX.1 functions. With careful implementation for infrequently used functions, there may be little or no performance or storage penalty, and the maintenance of already-standardized interfaces is a significant benefit.

2. Return the actual value computed by the function

   This technique can only be used with functions that return pointers to structures—routines that return character strings would have to wrap their output in an enclosing structure in order to return the output on the stack. There is also a negative performance impact inherent in this solution in that the output value has to be copied twice before it can be used by the calling function: once from the called routine's local buffers to the top of the stack, then from the top of the stack to the assignment target. Finally, many older compilers cannot support this technique due to a historical tendency to use internal static buffers to deliver the results of structure-valued functions.

3. Have the caller pass the address of a buffer to contain the computed value

   The only disadvantage of this approach is that extra arguments have to be provided by the calling program. It represents the most efficient solution to the problem, however, and, unlike the *malloc*( ) technique, it is semantically clear.

There are some routines (often groups of related routines) whose interfaces are inherently non-thread-safe because they communicate across multiple function invocations by means of static memory locations. The solution is to redesign the calls so that they are thread-safe, typically by passing the needed data as extra parameters. Unfortunately, this may require major changes to the interface as well.

A floating-point implementation using IEEE Std 754-1985 is a case in point. A less problematic example is the *rand48* family of pseudo-random number generators. The functions *getgrgid*( ), *getgrnam*( ), *getpwnam*( ), and *getpwuid*( ) are another such case.

The problems with *errno* are discussed in Alternative Solutions for Per-Thread errno (on page 3745).

Some functions can be thread-safe or not, depending on their arguments. These include the *tmpnam*( ) and *ctermid*( ) functions. These functions have pointers to character strings as arguments. If the pointers are not NULL, the functions store their results in the character string; however, if the pointers are NULL, the functions store their results in an area that may be static and thus subject to overwriting by successive calls. These should only be called by multi-thread applications when their arguments are non-NULL.

Asynchronous Safety and Thread-Safety

A floating-point implementation has many modes that effect rounding and other aspects of computation. Functions in some math library implementations may change the computation modes for the duration of a function call. If such a function call is interrupted by a signal or cancellation, the floating-point state is not required to be protected.

There is a significant cost to make floating-point operations async-cancel-safe or async-signal-safe; accordingly, neither form of async safety is required.

Functions Returning Pointers to Static Storage

For those functions that are not thread-safe because they return values in fixed size statically allocated structures, alternate ``_r'' forms are provided that pass a pointer to an explicit result structure. Those that return pointers into library-allocated buffers have forms provided with explicit buffer and length parameters.

For functions that return pointers to library-allocated buffers, it makes sense to provide ``_r'' versions that allow the application control over allocation of the storage in which results are returned. This allows the state used by these functions to be managed on an application-specific basis, supporting per-thread, per-process, or other application-specific sharing relationships.

Early proposals had provided ``_r'' versions for functions that returned pointers to variable-size buffers without providing a means for determining the required buffer size. This would have made using such functions exceedingly clumsy, potentially requiring iteratively calling them with increasingly larger guesses for the amount of storage required. Hence, *sysconf*( ) variables have been provided for such functions that return the maximum required buffer size.

Thus, the rule that has been followed by POSIX.1-2024 when adapting single-threaded non-thread-safe functions is as follows: all functions returning pointers to library-allocated storage should have ``_r'' versions provided, allowing the application control over the storage allocation. Those with variable-sized return values accept both a buffer address and a length parameter. The *sysconf*( ) variables are provided to supply the appropriate buffer sizes when

required. Implementors are encouraged to apply the same rule when adapting their own existing functions to a pthreads environment.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0020 [631], XSH/TC2-2008/0021 [826], and XSH/TC2-2008/0022 [631] are applied.

Austin Group Defect 188 is applied, removing *getenv*( ) from the list of functions that need not be thread-safe.

Austin Group Defect 696 is applied, requiring *readdir*( ) to be thread-safe except when concurrent calls are made for the same directory stream.

Austin Group Defect 922 is applied, adding the *secure_getenv*( ) function.

Austin Group Defect 1064 is applied, removing *basename*( ) and *dirname*( ) from the list of functions that need not be thread-safe.

Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899: 2018 standard.

B.2.9.2 Thread IDs

Separate applications should communicate through well-defined interfaces and should not depend on each other's implementation. For example, if a programmer decides to rewrite the *sort* utility using multiple threads, it should be easy to do this so that the interface to the *sort* utility does not change. Consider that if the user causes SIGINT to be generated while the *sort* utility is running, keeping the same interface means that the entire *sort* utility is killed, not just one of its threads. As another example, consider a realtime application that manages a reactor. Such an application may wish to allow other applications to control the priority at which it watches the control rods. One technique to accomplish this is to write the ID of the thread watching the control rods into a file and allow other programs to change the priority of that thread as they see fit. A simpler technique is to have the reactor process accept IPCs (Interprocess Communication messages) from other processes, telling it at a semantic level what priority the program should assign to watching the control rods. This allows the programmer greater flexibility in the implementation. For example, the programmer can change the implementation from having one thread per rod to having one thread watching all of the rods without changing the interface. Having threads live inside the process means that the implementation of a process is invisible to outside processes (excepting debuggers and system management tools).

Threads do not provide a protection boundary. Every thread model allows threads to share memory with other threads and encourages this sharing to be widespread. This means that one thread can wipe out memory that is needed for the correct functioning of other threads that are sharing its memory. Consequently, providing each thread with its own user and/or group IDs would not provide a protection boundary between threads sharing memory.

Some applications make the assumption that the implementation can always detect invalid uses of thread IDs of type **pthread_t**. This is an invalid assumption. Specifically, if **pthread_t** is defined as a pointer type, no access check needs to be performed before using the ID.

As with other interfaces that take pointer parameters, the outcome of passing an invalid parameter can result in an invalid memory reference or an attempt to access an undefined portion of a memory object, cause signals to be sent (SIGSEGV or SIGBUS) and possible termination of the process. This is a similar case to passing an invalid buffer pointer to *read*( ). Some implementations might implement *read*( ) as a system call and set an [EFAULT] error condition. Other implementations might contain parts of *read*( ) at user level and the first attempt to access data at an invalid reference will cause a signal to be sent instead.

If an implementation detects use of a thread ID after the end of its lifetime, it is recommended

that the function should fail and report an [ESRCH] error. This does not imply that implementations are required to return in this case. It is legitimate behavior to send an ``invalid memory reference'' signal (SIGSEGV or SIGBUS). It is the application's responsibility to use only valid thread IDs and to keep track of the lifetime of the underlying threads.

Austin Group Defect 792 is applied, clarifying thread lifetime.

Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

B.2.9.3 *Thread Mutexes*

Austin Group Defect 1216 is applied, adding *pthread_cond_clockwait*() and *pthread_mutex_clocklock*().

Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

B.2.9.4 *Thread Scheduling*

- Scheduling Implementation Models

  The following scheduling implementation models are presented in terms of threads and ``kernel entities''. This is to simplify exposition of the models, and it does not imply that an implementation actually has an identifiable ``kernel entity''.

  A kernel entity is not defined beyond the fact that it has scheduling attributes that are used to resolve contention with other kernel entities for execution resources. A kernel entity may be thought of as an envelope that holds a thread or a separate kernel thread. It is not a conventional process, although it shares with the process the attribute that it has a single thread of control; it does not necessarily imply an address space, open files, and so on. It is better thought of as a primitive facility upon which conventional processes and threads may be constructed.

  — System Thread Scheduling Model

    This model consists of one thread per kernel entity. The kernel entity is solely responsible for scheduling thread execution on one or more processors. This model schedules all threads against all other threads in the system using the scheduling attributes of the thread.

  — Process Scheduling Model

    A generalized process scheduling model consists of two levels of scheduling. A threads library creates a pool of kernel entities, as required, and schedules threads to run on them using the scheduling attributes of the threads. Typically, the size of the pool is a function of the simultaneously runnable threads, not the total number of threads. The kernel then schedules the kernel entities onto processors according to their scheduling attributes, which are managed by the threads library. This set model potentially allows a wide range of mappings between threads and kernel entities.

- System and Process Scheduling Model Performance

  There are a number of important implications on the performance of applications using these scheduling models. The process scheduling model potentially provides lower overhead for making scheduling decisions, since there is no need to access kernel-level information or functions and the set of schedulable entities is smaller (only the threads within the process).

  On the other hand, since the kernel is also making scheduling decisions regarding the

132301 system resources under its control (for example, CPU(s), I/O devices, memory), decisions
132302 that do not take thread scheduling parameters into account can result in unspecified
132303 delays for realtime application threads, causing them to miss maximum response time
132304 limits.

132305 • Rate Monotonic Scheduling

132306 Rate monotonic scheduling was considered, but rejected for standardization in the context
132307 of pthreads. A sporadic server policy is included.

132308 • Scheduling Options

132309 In POSIX.1-2024, the basic thread scheduling functions are defined under the threads
132310 functionality, so that they are required of all threads implementations. However, there are
132311 no specific scheduling policies required by this functionality to allow for conforming
132312 thread implementations that are not targeted to realtime applications.

132313 Specific standard scheduling policies are defined to be under the Thread Execution
132314 Scheduling option, and they are specifically designed to support realtime applications by
132315 providing predictable resource-sharing sequences. The name of this option was chosen to
132316 emphasize that this functionality is defined as appropriate for realtime applications that
132317 require simple priority-based scheduling.

132318 It is recognized that these policies are not necessarily satisfactory for some multi-processor
132319 implementations, and work is ongoing to address a wider range of scheduling behaviors.
132320 The interfaces have been chosen to create abundant opportunity for future scheduling
132321 policies to be implemented and standardized based on this interface. In order to
132322 standardize a new scheduling policy, all that is required (from the standpoint of thread
132323 scheduling attributes) is to define a new policy name, new members of the thread
132324 attributes object, and functions to set these members when the scheduling policy is equal
132325 to the new value.

132326 **Scheduling Contention Scope**

132327 In order to accommodate the requirement for realtime response, each thread has a scheduling
132328 contention scope attribute. Threads with a system scheduling contention scope have to be
132329 scheduled with respect to all other threads in the system. These threads are usually bound to a
132330 single kernel entity that reflects their scheduling attributes and are directly scheduled by the
132331 kernel.

132332 Threads with a process scheduling contention scope need be scheduled only with respect to the
132333 other threads in the process. These threads may be scheduled within the process onto a pool of
132334 kernel entities. The implementation is also free to bind these threads directly to kernel entities
132335 and let them be scheduled by the kernel. Process scheduling contention scope allows the
132336 implementation the most flexibility and is the default if both contention scopes are supported
132337 and none is specified.

132338 Thus, the choice by implementors to provide one or the other (or both) of these scheduling
132339 models is driven by the need of their supported application domains for worst-case (that is,
132340 realtime) response, or average-case (non-realtime) response.

Scheduling Allocation Domain

The SCHED_FIFO and SCHED_RR scheduling policies take on different characteristics on a multi-processor. Other scheduling policies are also subject to changed behavior when executed on a multi-processor. The concept of scheduling allocation domain determines the set of processors on which the threads of an application may run. By considering the application's processor scheduling allocation domain for its threads, scheduling policies can be defined in terms of their behavior for varying processor scheduling allocation domain values. It is conceivable that not all scheduling allocation domain sizes make sense for all scheduling policies on all implementations. The concept of scheduling allocation domain, however, is a useful tool for the description of multi-processor scheduling policies.

The ``process control'' approach to scheduling obtains significant performance advantages from dynamic scheduling allocation domain sizes when it is applicable.

Non-Uniform Memory Access (NUMA) multi-processors may use a system scheduling structure that involves reassignment of threads among scheduling allocation domains. In NUMA machines, a natural model of scheduling is to match scheduling allocation domains to clusters of processors. Load balancing in such an environment requires changing the scheduling allocation domain to which a thread is assigned.

Scheduling Documentation

Implementation-provided scheduling policies need to be completely documented in order to be useful. This documentation includes a description of the attributes required for the policy, the scheduling interaction of threads running under this policy and all other supported policies, and the effects of all possible values for processor scheduling allocation domain. Note that for the implementor wishing to be minimally-compliant, it is (minimally) acceptable to define the behavior as undefined.

Scheduling Contention Scope Attribute

The scheduling contention scope defines how threads compete for resources. Within POSIX.1-2024, scheduling contention scope is used to describe only how threads are scheduled in relation to one another in the system. That is, either they are scheduled against all other threads in the system (``system scope'') or only against those threads in the process (``process scope''). In fact, scheduling contention scope may apply to additional resources, including virtual timers and profiling, which are not currently considered by POSIX.1-2024.

Mixed Scopes

If only one scheduling contention scope is supported, the scheduling decision is straightforward. To perform the processor scheduling decision in a mixed scope environment, it is necessary to map the scheduling attributes of the thread with process-wide contention scope to the same attribute space as the thread with system-wide contention scope.

Since a conforming implementation has to support one and may support both scopes, it is useful to discuss the effects of such choices with respect to example applications. If an implementation supports both scopes, mixing scopes provides a means of better managing system-level (that is, kernel-level) and library-level resources. In general, threads with system scope will require the resources of a separate kernel entity in order to guarantee the scheduling semantics. On the other hand, threads with process scope can share the resources of a kernel entity while maintaining the scheduling semantics.

The application is free to create threads with dedicated kernel resources, and other threads that multiplex kernel resources. Consider the example of a window server. The server allocates two threads per widget: one thread manages the widget user interface (including drawing), while

the other thread takes any required application action. This allows the widget to be ``active'' while the application is computing. A screen image may be built from thousands of widgets. If each of these threads had been created with system scope, then most of the kernel-level resources might be wasted, since only a few widgets are active at any one time. In addition, mixed scope is particularly useful in a window server where one thread with high priority and system scope handles the mouse so that it tracks well. As another example, consider a database server. For each of the hundreds or thousands of clients supported by a large server, an equivalent number of threads will have to be created. If each of these threads were system scope, the consequences would be the same as for the window server example above. However, the server could be constructed so that actual retrieval of data is done by several dedicated threads. Dedicated threads that do work for all clients frequently justify the added expense of system scope. If it were not permissible to mix system and process threads in the same process, this type of solution would not be possible.

Dynamic Thread Scheduling Parameters Access

In many time-constrained applications, there is no need to change the scheduling attributes dynamically during thread or process execution, since the general use of these attributes is to reflect directly the time constraints of the application. Since these time constraints are generally imposed to meet higher-level system requirements, such as accuracy or availability, they frequently should remain unchanged during application execution.

However, there are important situations in which the scheduling attributes should be changed. Generally, this will occur when external environmental conditions exist in which the time constraints change. Consider, for example, a space vehicle major mode change, such as the change from ascent to descent mode, or the change from the space environment to the atmospheric environment. In such cases, the frequency with which many of the sensors or actuators need to be read or written will change, which will necessitate a priority change. In other cases, even the existence of a time constraint might be temporary, necessitating not just a priority change, but also a policy change for ongoing threads or processes. For this reason, it is critical that the interface should provide functions to change the scheduling parameters dynamically, but, as with many of the other realtime functions, it is important that applications use them properly to avoid the possibility of unnecessarily degrading performance.

In providing functions for dynamically changing the scheduling behavior of threads, there were two options: provide functions to get and set the individual scheduling parameters of threads, or provide a single interface to get and set all the scheduling parameters for a given thread simultaneously. Both approaches have merit. Access functions for individual parameters allow simpler control of thread scheduling for simple thread scheduling parameters. However, a single function for setting all the parameters for a given scheduling policy is required when first setting that scheduling policy. Since the single all-encompassing functions are required, it was decided to leave the interface as minimal as possible. Note that simpler functions (such as *pthread_setprio*( ) for threads running under the priority-based schedulers) can be easily defined in terms of the all-encompassing functions.

If the *pthread_setschedparam*( ) function executes successfully, it will have set all of the scheduling parameter values indicated in *param*; otherwise, none of the scheduling parameters will have been modified. This is necessary to ensure that the scheduling of this and all other threads continues to be consistent in the presence of an erroneous scheduling parameter.

The [EPERM] error value is included in the list of possible *pthread_setschedparam*( ) error returns as a reflection of the fact that the ability to change scheduling parameters increases risks to the implementation and application performance if the scheduling parameters are changed improperly. For this reason, and based on some existing practice, it was felt that some implementations would probably choose to define specific permissions for changing either a

thread's own or another thread's scheduling parameters. POSIX.1-2024 does not include portable methods for setting or retrieving permissions, so any such use of permissions is completely unspecified.

Mutex Initialization Scheduling Attributes

In a priority-driven environment, a direct use of traditional primitives like mutexes and condition variables can lead to unbounded priority inversion, where a higher priority thread can be blocked by a lower priority thread, or set of threads, for an unbounded duration of time. As a result, it becomes impossible to guarantee thread deadlines. Priority inversion can be bounded and minimized by the use of priority inheritance protocols. This allows thread deadlines to be guaranteed even in the presence of synchronization requirements.

Two useful but simple members of the family of priority inheritance protocols are the basic priority inheritance protocol and the priority ceiling protocol emulation. Under the Basic Priority Inheritance protocol (governed by the Non-Robust Mutex Priority Inheritance option), a thread that is blocking higher priority threads executes at the priority of the highest priority thread that it blocks. This simple mechanism allows priority inversion to be bounded by the duration of critical sections and makes timing analysis possible.

Under the Priority Ceiling Protocol Emulation protocol (governed by the Thread Priority Protection option), each mutex has a priority ceiling, usually defined as the priority of the highest priority thread that can lock the mutex. When a thread is executing inside critical sections, its priority is unconditionally increased to the highest of the priority ceilings of all the mutexes owned by the thread. This protocol has two very desirable properties in uni-processor systems. First, a thread can be blocked by a lower priority thread for at most the duration of one single critical section. Furthermore, when the protocol is correctly used in a single processor, and if threads do not become blocked while owning mutexes, mutual deadlocks are prevented.

The priority ceiling emulation can be extended to multiple processor environments, in which case the values of the priority ceilings will be assigned depending on the kind of mutex that is being used: local to only one processor, or global, shared by several processors. Local priority ceilings will be assigned the usual way, equal to the priority of the highest priority thread that may lock that mutex. Global priority ceilings will usually be assigned a priority level higher than all the priorities assigned to any of the threads that reside in the involved processors to avoid the effect called remote blocking.

Change the Priority Ceiling of a Mutex

In order for the priority protect protocol to exhibit its desired properties of bounding priority inversion and avoidance of deadlock, it is critical that the ceiling priority of a mutex be the same as the priority of the highest thread that can ever hold it, or higher. Thus, if the priorities of the threads using such mutexes never change dynamically, there is no need ever to change the priority ceiling of a mutex.

However, if a major system mode change results in an altered response time requirement for one or more application threads, their priority has to change to reflect it. It will occasionally be the case that the priority ceilings of mutexes held also need to change. While changing priority ceilings should generally be avoided, it is important that POSIX.1-2024 provide these interfaces for those cases in which it is necessary.

B.2.9.5 Thread Cancellation

Many existing threads packages have facilities for canceling an operation or canceling a thread. These facilities are used for implementing user requests (such as the CANCEL button in a window-based application), for implementing OR parallelism (for example, telling the other threads to stop working once one thread has found a forced mate in a parallel chess program), or for implementing the ABORT mechanism in Ada.

POSIX programs traditionally have used the signal mechanism combined with either *longjmp*() or polling to cancel operations. Many POSIX programmers have trouble using these facilities to solve their problems efficiently in a single-threaded process. With the introduction of threads, these solutions become even more difficult to use.

The main issues with implementing a cancellation facility are specifying the operation to be canceled, cleanly releasing any resources allocated to that operation, controlling when the target notices that it has been canceled, and defining the interaction between asynchronous signals and cancellation.

Specifying the Operation to Cancel

Consider a thread that calls through five distinct levels of program abstraction and then, inside the lowest-level abstraction, calls a function that suspends the thread. (An abstraction boundary is a layer at which the client of the abstraction sees only the service being provided and can remain ignorant of the implementation. Abstractions are often layered, each level of abstraction being a client of the lower-level abstraction and implementing a higher-level abstraction.) Depending on the semantics of each abstraction, one could imagine wanting to cancel only the call that causes suspension, only the bottom two levels, or the operation being done by the entire thread. Canceling operations at a finer grain than the entire thread is difficult because threads are active and they may be run in parallel on a multi-processor. By the time one thread can make a request to cancel an operation, the thread performing the operation may have completed that operation and gone on to start another operation whose cancellation is not desired. Thread IDs are not reused until the thread has exited, and either it was created with the *Attr detachstate* attribute set to PTHREAD_CREATE_DETACHED or the *pthread_join*() or *pthread_detach*() function has been called for that thread. Consequently, a thread cancellation will never be misdirected when the thread terminates. For these reasons, the canceling of operations is done at the granularity of the thread. Threads are designed to be inexpensive enough so that a separate thread may be created to perform each separately cancelable operation; for example, each possibly long running user request.

For cancellation to be used in existing code, cancellation scopes and handlers will have to be established for code that needs to release resources upon cancellation, so that it follows the programming discipline described in the text.

A Special Signal Versus a Special Interface

Two different mechanisms were considered for providing the cancellation interfaces. The first was to provide an interface to direct signals at a thread and then to define a special signal that had the required semantics. The other alternative was to use a special interface that delivered the correct semantics to the target thread.

The solution using signals produced a number of problems. It required the implementation to provide cancellation in terms of signals whereas a perfectly valid (and possibly more efficient) implementation could have both layered on a low-level set of primitives. There were so many exceptions to the special signal (it cannot be used with *kill*(), no POSIX.1 interfaces can be used with it) that it was clearly not a valid signal. Its semantics on delivery were also completely different from any existing POSIX.1 signal. As such, a special interface that did not mandate the implementation and did not confuse the semantics of signals and cancellation was felt to be the

132526 better solution.

132527 **Races Between Cancellation and Resuming Execution**

132528 Due to the nature of cancellation, there is generally no synchronization between the thread
132529 requesting the cancellation of a blocked thread and events that may cause that thread to resume
132530 execution. For this reason, and because excess serialization hurts performance, when both an
132531 event that a thread is waiting for has occurred and a cancellation request has been made and
132532 cancellation is enabled, POSIX.1-2024 explicitly allows the implementation to choose between
132533 returning from the blocking call or acting on the cancellation request.

132534 **Interaction of Cancellation with Asynchronous Signals**

132535 A typical use of cancellation is to acquire a lock on some resource and to establish a cancellation
132536 cleanup handler for releasing the resource when and if the thread is canceled.

132537 A correct and complete implementation of cancellation in the presence of asynchronous signals
132538 requires considerable care. An implementation has to push a cancellation cleanup handler on the
132539 cancellation cleanup stack while maintaining the integrity of the stack data structure. If an
132540 asynchronously-generated signal is posted to the thread during a stack operation, the signal
132541 handler cannot manipulate the cancellation cleanup stack. As a consequence, asynchronous
132542 signal handlers may not cancel threads or otherwise manipulate the cancellation state of a
132543 thread. Threads may, of course, be canceled by another thread that used a *sigwait*() function to
132544 wait synchronously for an asynchronous signal.

132545 In order for cancellation to function correctly, it is required that asynchronous signal handlers
132546 not change the cancellation state. This requires that some elements of existing practice, such as
132547 using *longjmp*() to exit from an asynchronous signal handler implicitly, be prohibited in cases
132548 where the integrity of the cancellation state of the interrupt thread cannot be ensured.

132549 **Thread Cancellation Overview**

132550 • Cancelability States

132551 The three possible cancelability states (disabled, deferred, and asynchronous) are encoded
132552 into two separate bits ((disable, enable) and (deferred, asynchronous)) to allow them to be
132553 changed and restored independently. For instance, short code sequences that will not block
132554 sometimes disable cancelability on entry and restore the previous state upon exit.
132555 Likewise, long or unbounded code sequences containing no convenient explicit
132556 cancellation points will sometimes set the cancelability type to asynchronous on entry and
132557 restore the previous value upon exit.

132558 • Cancellation Points

132559 Cancellation points are points inside of certain functions where a thread has to act on any
132560 pending cancellation request when cancelability is enabled. For functions in the ``shall
132561 occur'' list, a cancellation check must be performed on every call regardless of whether,
132562 absent the cancellation, the call would have blocked. For functions in the ``may occur'' list,
132563 a cancellation check may be performed on some calls but not others; i.e., whether or not a
132564 cancellation point occurs when one of these functions is being executed can depend on
132565 current conditions.

132566 The idea was considered of allowing implementations to define whether blocking calls
132567 such as *read*() should be cancellation points. It was decided that it would adversely affect
132568 the design of conforming applications if blocking calls were not cancellation points
132569 because threads could be left blocked in an uncancelable state.

132570 There are several important blocking routines that are specifically not made cancellation

132571 points:

132572 — *pthread_mutex_lock*( )

132573 If *pthread_mutex_lock*( ) were a cancellation point, every routine that called it would
132574 also become a cancellation point (that is, any routine that touched shared state would
132575 automatically become a cancellation point). For example, *malloc*( ), *free*( ), and *rand*( )
132576 would become cancellation points under this scheme. Having too many cancellation
132577 points makes programming very difficult, leading to either much disabling and
132578 restoring of cancelability or much difficulty in trying to arrange for reliable cleanup
132579 at every possible place.

132580 Since *pthread_mutex_lock*( ) is not a cancellation point, threads could result in being
132581 blocked uninterruptibly for long periods of time if mutexes were used as a general
132582 synchronization mechanism. As this is normally not acceptable, mutexes should only
132583 be used to protect resources that are held for small fixed lengths of time where not
132584 being able to be canceled will not be a problem. Resources that need to be held
132585 exclusively for long periods of time should be protected with condition variables.

132586 — *pthread_barrier_wait*( )

132587 Canceling a barrier wait will render a barrier unusable. Similar to a barrier timeout
132588 (which the standard developers rejected), there is no way to guarantee the
132589 consistency of a barrier's internal data structures if a barrier wait is canceled.

132590 — *pthread_spin_lock*( )

132591 As with mutexes, spin locks should only be used to protect resources that are held for
132592 small fixed lengths of time where not being cancelable will not be a problem.

132593 Every library routine should specify whether or not it includes any cancellation points.
132594 Typically, only those routines that may block or compute indefinitely need to include
132595 cancellation points.

132596 Correctly coded routines only reach cancellation points after having set up a cancellation
132597 cleanup handler to restore invariants if the thread is canceled at that point. Being
132598 cancelable only at specified cancellation points allows programmers to keep track of
132599 actions needed in a cancellation cleanup handler more easily. A thread should only be
132600 made asynchronously cancelable when it is not in the process of acquiring or releasing
132601 resources or otherwise in a state from which it would be difficult or impossible to recover.

132602 • Thread Cancellation Cleanup Handlers

132603 The cancellation cleanup handlers provide a portable mechanism, easy to implement, for
132604 releasing resources and restoring invariants. They are easier to use than signal handlers
132605 because they provide a stack of cancellation cleanup handlers rather than a single handler,
132606 and because they have an argument that can be used to pass context information to the
132607 handler.

132608 The alternative to providing these simple cancellation cleanup handlers (whose only use is
132609 for cleaning up when a thread is canceled) is to define a general exception package that
132610 could be used for handling and cleaning up after hardware traps and software-detected
132611 errors. This was too far removed from the charter of providing threads to handle
132612 asynchrony. However, it is an explicit goal of POSIX.1-2024 to be compatible with existing
132613 exception facilities and languages having exceptions.

132614 The interaction of this facility and other procedure-based or language-level exception
132615 facilities is unspecified in this version of POSIX.1-2024. However, it is intended that it be
132616 possible for an implementation to define the relationship between these cancellation

cleanup handlers and Ada, C++, or other language-level exception handling facilities.

It was suggested that the cancellation cleanup handlers should also be called when the process exits or calls the *exec* function. This was rejected partly due to the performance problem caused by having to call the cancellation cleanup handlers of every thread before the operation could continue. The other reason was that the only state expected to be cleaned up by the cancellation cleanup handlers would be the intraprocess state. Any handlers that are to clean up the interprocess state would be registered with *atexit*(). There is the orthogonal problem that the *exec* functions do not honor the *atexit*() handlers, but resolving this is beyond the scope of POSIX.1-2024.

- Async-Cancel Safety

  A function is said to be async-cancel-safe if it is written in such a way that entering the function with asynchronous cancelability enabled will not cause any invariants to be violated, even if a cancellation request is delivered at any arbitrary instruction. Functions that are async-cancel-safe are often written in such a way that they need to acquire no resources for their operation and the visible variables that they may write are strictly limited.

  Any routine that gets a resource as a side-effect cannot be made async-cancel-safe (for example, *malloc*()). If such a routine were called with asynchronous cancelability enabled, it might acquire the resource successfully, but as it was returning to the client, it could act on a cancellation request. In such a case, the application would have no way of knowing whether the resource was acquired or not.

  Indeed, because many interesting routines cannot be made async-cancel-safe, most library routines in general are not async-cancel-safe. Every library routine should specify whether or not it is async-cancel safe so that programmers know which routines can be called from code that is asynchronously cancelable.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/8 is applied, adding the *pselect*() function to the list of functions with cancellation points.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/5 is applied, adding the *fdatasync*() function into the table of functions that shall have cancellation points.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/6 is applied, adding the numerous functions into the table of functions that may have cancellation points.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/7 is applied, clarifying the requirements in Thread Cancellation Cleanup Handlers.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0023 [627], XSH/TC2-2008/0024 [627,632], XSH/TC2-2008/0025 [627], XSH/TC2-2008/0026 [632], and XSH/TC2-2008/0027 [622] are applied.

Austin Group Defect 411 is applied, adding *accept4*() to the table of functions that shall have cancellation points.

Austin Group Defect 508 is applied, adding *ptsname*() and *ptsname_r*() to the table of functions that may have cancellation points.

Austin Group Defect 614 is applied, adding *posix_close*() to the table of functions that shall have cancellation points.

Austin Group Defect 697 is applied, adding *posix_getdents*() to the table of functions that may have cancellation points.

Austin Group Defect 729 is applied, adding *posix_devctl*() to the table of functions that may have

cancellation points.

Austin Group Defect 841 is applied, allowing *pthread_setcancelstate*() to be used to disable cancellation in a signal catching function in order to avoid undefined behavior when the signal is delivered during execution of a function that is not async-cancel-safe.

Austin Group Defect 1076 is applied, moving *sem_wait*() and *sem_timedwait*() from the table of functions that are required to have cancellation points to the table of functions that may have cancellation points.

Austin Group Defect 1122 is applied, adding *bindtextdomain*() and the *gettext* family of functions to the table of functions that may have cancellation points.

Austin Group Defect 1143 is applied, clarifying the conditions under which it is unspecified whether the cancellation request is acted upon or whether the cancellation request remains pending.

Austin Group Defect 1216 is applied, adding *pthread_cond_clockwait*() to the table of functions that are required to have cancellation points, and adding *pthread_rwlock_clockwrlock*(), *pthread_rwlock_clockrdlock*(), and *sem_clockwait*() to the table of functions that may have cancellation points.

Austin Group Defect 1263 is applied, adding *ppoll*() to the table of functions that are required to have cancellation points.

Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

Austin Group Defect 1410 is applied, removing the *asctime_r*() and *ctime_r*() functions.

B.2.9.6 Thread Read-Write Locks

Background

Read-write locks are often used to allow parallel access to data on multi-processors, to avoid context switches on uni-processors when multiple threads access the same data, and to protect data structures that are frequently accessed (that is, read) but rarely updated (that is, written). The in-core representation of a file system directory is a good example of such a data structure. One would like to achieve as much concurrency as possible when searching directories, but limit concurrent access when adding or deleting files.

Although read-write locks can be implemented with mutexes and condition variables, such implementations are significantly less efficient than is possible. Therefore, this synchronization primitive is included in POSIX.1-2024 for the purpose of allowing more efficient implementations in multi-processor systems.

Queuing of Waiting Threads

The *pthread_rwlock_unlock*() function description states that one writer or one or more readers must acquire the lock if it is no longer held by any thread as a result of the call. However, the function does not specify which thread(s) acquire the lock, unless the Thread Execution Scheduling option is supported.

The standard developers considered the issue of scheduling with respect to the queuing of threads blocked on a read-write lock. The question turned out to be whether POSIX.1-2024 should require priority scheduling of read-write locks for threads whose execution scheduling policy is priority-based (for example, SCHED_FIFO or SCHED_RR). There are tradeoffs between priority scheduling, the amount of concurrency achievable among readers, and the prevention of writer and/or reader starvation.

For example, suppose one or more readers hold a read-write lock and the following threads request the lock in the listed order:

```
pthread_rwlock_wrlock() - Low priority thread writer_a
pthread_rwlock_rdlock() - High priority thread reader_a
pthread_rwlock_rdlock() - High priority thread reader_b
pthread_rwlock_rdlock() - High priority thread reader_c
```

When the lock becomes available, should *writer_a* block the high priority readers? Or, suppose a read-write lock becomes available and the following are queued:

```
pthread_rwlock_rdlock() - Low priority thread reader_a
pthread_rwlock_rdlock() - Low priority thread reader_b
pthread_rwlock_rdlock() - Low priority thread reader_c
pthread_rwlock_wrlock() - Medium priority thread writer_a
pthread_rwlock_rdlock() - High priority thread reader_d
```

If priority scheduling is applied then *reader_d* would acquire the lock and *writer_a* would block the remaining readers. But should the remaining readers also acquire the lock to increase concurrency? The solution adopted takes into account that when the Thread Execution Scheduling option is supported, high priority threads may in fact starve low priority threads (the application developer is responsible in this case for designing the system in such a way that this starvation is avoided). Therefore, POSIX.1-2024 specifies that high priority readers take precedence over lower priority writers. However, to prevent writer starvation from threads of the same or lower priority, writers take precedence over readers of the same or lower priority.

Priority inheritance mechanisms are non-trivial in the context of read-write locks. When a high priority writer is forced to wait for multiple readers, for example, it is not clear which subset of the readers should inherit the writer's priority. Furthermore, the internal data structures that record the inheritance must be accessible to all readers, and this implies some sort of serialization that could negate any gain in parallelism achieved through the use of multiple readers in the first place. Finally, existing practice does not support the use of priority inheritance for read-write locks. Therefore, no specification of priority inheritance or priority ceiling is attempted. If reliable priority-scheduled synchronization is absolutely required, it can always be obtained through the use of mutexes.

Comparison to fcntl() Locks

The read-write locks and the *fcntl*() locks in POSIX.1-2024 share a common goal: increasing concurrency among readers, thus increasing throughput and decreasing delay.

However, the read-write locks have two features not present in the *fcntl*() locks. First, under priority scheduling, read-write locks are granted in priority order. Second, also under priority scheduling, writer starvation is prevented by giving writers preference over readers of equal or lower priority.

Also, read-write locks can be used in systems lacking a file system, such as those conforming to the minimal realtime system profile of IEEE Std 1003.13-1998.

History of Resolution Issues

Based upon some balloting objections, early drafts specified the behavior of threads waiting on a read-write lock during the execution of a signal handler, as if the thread had not called the lock operation. However, this specified behavior would require implementations to establish internal signal handlers even though this situation would be rare, or never happen for many programs. This would introduce an unacceptable performance hit in comparison to the little additional functionality gained. Therefore, the behavior of read-write locks and signals was reverted back to its previous mutex-like specification.

B.2.9.7 *Thread Interactions with File Operations*

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0028 [498] is applied.

Austin Group Defect 411 is applied, adding *dup3*( ).

Austin Group Defect 695 is applied, extending the requirements in this section to non-regular files.

B.2.9.8 *Use of Application-Managed Thread Stacks*

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/8 is applied, adding this new section. It was added to make it clear that the current standard does not allow an application to determine when a stack can be reclaimed. This may be addressed in a future version.

B.2.9.9 *Synchronization Object Copies and Alternative Mappings*

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0029 [972] is applied.

Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

B.2.10 Sockets

The base document for the sockets interfaces in POSIX.1-2024 is the XNS, Issue 5.2 specification. This was primarily chosen as it aligns with IPv6. Additional material has been added from IEEE Std 1003.1g-2000, notably socket concepts, raw sockets, the *pselect*( ) function, the *sockatmark*( ) function, and the **<sys/select.h>** header.

B.2.10.1 *Address Families*

There is no additional rationale provided for this section.

B.2.10.2 *Addressing*

There is no additional rationale provided for this section.

B.2.10.3 *Protocols*

There is no additional rationale provided for this section.

132775 *B.2.10.4 Routing*

132776 There is no additional rationale provided for this section.

132777 *B.2.10.5 Interfaces*

132778 There is no additional rationale provided for this section.

132779 *B.2.10.6 Socket Types*

132780 The type **socklen_t** was invented to cover the range of implementations seen in the field. The
132781 intent of **socklen_t** is to be the type for all lengths that are naturally bounded in size; that is, that
132782 they are the length of a buffer which cannot sensibly become of massive size: network addresses,
132783 host names, string representations of these, ancillary data, control messages, and socket options
132784 are examples. Truly boundless sizes are represented by **size_t** as in *read*( ), *write*( ), and so on.

132785 All **socklen_t** types were originally (in BSD UNIX) of type **int**. During the development of
132786 POSIX.1-2024, it was decided to change all buffer lengths to **size_t**, which appears at face value
132787 to make sense. When dual mode 32/64-bit systems came along, this choice unnecessarily
132788 complicated system interfaces because **size_t** (with **long**) was a different size under ILP32 and
132789 LP64 models. Reverting to **int** would have happened except that some implementations had
132790 already shipped 64-bit-only interfaces. The compromise was a type which could be defined to be
132791 any size by the implementation: **socklen_t**.

132792 *B.2.10.7 Socket I/O Mode*

132793 There is no additional rationale provided for this section.

132794 *B.2.10.8 Socket Owner*

132795 There is no additional rationale provided for this section.

132796 *B.2.10.9 Socket Queue Limits*

132797 There is no additional rationale provided for this section.

132798 *B.2.10.10 Pending Error*

132799 There is no additional rationale provided for this section.

132800 *B.2.10.11 Socket Receive Queue*

132801 There is no additional rationale provided for this section.

132802 *B.2.10.12 Socket Out-of-Band Data State*

132803 There is no additional rationale provided for this section.

132804 *B.2.10.13 Connection Indication Queue*

132805 There is no additional rationale provided for this section.

132806 *B.2.10.14 Signals*

132807 There is no additional rationale provided for this section.

132808 *B.2.10.15 Asynchronous Errors*

132809 Austin Group Defect 1010 is applied, removing [EHOSTDOWN] from the list of asynchronous
132810 errors.

132811 *B.2.10.16 Use of Options*

132812 Austin Group Defect 840 is applied, adding SO_DOMAIN and SO_PROTOCOL.

132813 Austin Group Defect 1337 is applied, clarifying socket option default values.

132814 *B.2.10.17 Use of Sockets for Local UNIX Connections*

132815 There is no additional rationale provided for this section.

132816 *B.2.10.18 Use of Sockets over Internet Protocols*

132817 A raw socket allows privileged users direct access to a protocol; for example, raw access to the
132818 IP and ICMP protocols is possible through raw sockets. Raw sockets are intended for
132819 knowledgeable applications that wish to take advantage of some protocol feature not directly
132820 accessible through the other sockets interfaces.

132821 *B.2.10.19 Use of Sockets over Internet Protocols Based on IPv4*

132822 There is no additional rationale provided for this section.

132823 *B.2.10.20 Use of Sockets over Internet Protocols Based on IPv6*

132824 The Open Group Base Resolution bwg2001-012 is applied, clarifying that IPv6 implementations
132825 are required to support use of AF_INET6 sockets over IPv4.

132826 Austin Group Defect 411 is applied, adding *accept4*().

132827 B.2.11 Data Types

132828 *B.2.11.1 Defined Types*

132829 The requirement that additional types defined in this section end in ``_t'' was prompted by the
132830 problem of name space pollution. It is difficult to define a type (where that type is not one
132831 defined by POSIX.1-2024) in one header file and use it in another without adding symbols to the
132832 name space of the program. To allow implementors to provide their own types, all conforming
132833 applications are required to avoid symbols ending in ``_t'', which permits the implementor to

provide additional types. Because a major use of types is in the definition of structure members, which can (and in many cases must) be added to the structures defined in POSIX.1-2024, the need for additional types is compelling.

The types, such as **ushort** and **ulong**, which are in common usage, are not defined in POSIX.1-2024 (although **ushort_t** would be permitted as an extension). They can be added to **<sys/types.h>** using a feature test macro (see Section B.2.2.1, on page 3737). A suggested symbol for these is _SYSIII. Similarly, the types like **u_short** would probably be best controlled by _BSD.

Some of these symbols may appear in other headers; see Section B.2.2.2 (on page 3738).

dev_t This type may be made large enough to accommodate host-locality considerations of networked systems.

This type must be arithmetic. Earlier proposals allowed this to be non-arithmetic (such as a structure) and provided a *samefile*( ) function for comparison.

gid_t Some implementations had separated **gid_t** from **uid_t** before POSIX.1 was completed. It would be difficult for them to coalesce them when it was unnecessary. Additionally, it is quite possible that user IDs might be different from group IDs because the user ID might wish to span a heterogeneous network, where the group ID might not.

For current implementations, the cost of having a separate **gid_t** will be only lexical.

mode_t This type was chosen so that implementations could choose the appropriate integer type, and for compatibility with the ISO C standard. 4.3 BSD uses **unsigned short** and the SVID uses **ushort**, which is the same. Historically, only the low-order sixteen bits are significant.

nlink_t This type was introduced in place of **short** for *st_nlink* (see the **<sys/stat.h>** header) in response to an objection that **short** was too small.

off_t This type is used to represent a file offset or file size. On systems supporting large files, **off_t** is larger than 32 bits in at least one programming environment. Other programming environments may use different sizes for **off_t**, for compatibility or other reasons.

pid_t The inclusion of this symbol was controversial because it is tied to the issue of the representation of a process ID as a number. From the point of view of a conforming application, process IDs should be ``magic cookies''[9] that are produced by calls such as *fork*( ), used by calls such as *waitpid*( ) or *kill*( ), and not otherwise analyzed (except that the sign is used as a flag for certain operations).

The concept of a {PID_MAX} value interacted with this in early proposals. Treating process IDs as an opaque type both removes the requirement for {PID_MAX} and allows systems to be more flexible in providing process IDs that span a large range of values, or a small one.

Since the values in **uid_t**, **gid_t**, and **pid_t** will be numbers generally, and potentially both large in magnitude and sparse, applications that are based on arrays of objects of this type are unlikely to be fully portable in any case. Solutions that treat them as magic cookies will be portable.

---

9. An historical term meaning: ``An opaque object, or token, of determinate size, whose significance is known only to the entity which created it. An entity receiving such a token from the generating entity may only make such use of the `cookie' as is defined and permitted by the supplying entity.''

{CHILD_MAX} precludes the possibility of a ``toy implementation'', where there would only be one process.

ssize_t This is intended to be a signed analog of **size_t**. The wording is such that an implementation may either choose to use a longer type or simply to use the signed version of the type that underlies **size_t**. All functions that return **ssize_t** (*read*( ) and *write*( )) describe as ``implementation-defined'' the result of an input exceeding {SSIZE_MAX}. It is recognized that some implementations might have **int**s that are smaller than **size_t**. A conforming application would be constrained not to perform I/O in pieces larger than {SSIZE_MAX}, but a conforming application using extensions would be able to use the full range if the implementation provided an extended range, while still having a single type-compatible interface.

The symbols **size_t** and **ssize_t** are also required in **<unistd.h>** to minimize the changes needed for calls to *read*( ) and *write*( ). Implementors are reminded that it must be possible to include both **<sys/types.h>** and **<unistd.h>** in the same program (in either order) without error.

uid_t Before the addition of this type, the data types used to represent these values varied throughout early proposals. The **<sys/stat.h>** header defined these values as type **short**, the **<passwd.h>** file (now **<pwd.h>** and **<grp.h>**) used an **int**, and *getuid*( ) returned an **int**. In response to a strong objection to the inconsistent definitions, all the types were switched to **uid_t**.

In practice, those historical implementations that use varying types of this sort can typedef **uid_t** to **short** with no serious consequences.

The problem associated with this change concerns object compatibility after structure size changes. Since most implementations will define **uid_t** as a short, the only substantive change will be a reduction in the size of the **passwd** structure. Consequently, implementations with an overriding concern for object compatibility can pad the structure back to its current size. For that reason, this problem was not considered critical enough to warrant the addition of a separate type to POSIX.1.

The types **uid_t** and **gid_t** are magic cookies. There is no {UID_MAX} defined by POSIX.1, and no structure imposed on **uid_t** and **gid_t** other than that they be positive arithmetic types. (In fact, they could be **unsigned char**.) There is no maximum or minimum specified for the number of distinct user or group IDs.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0030 [733] is applied.

Austin Group Defect 697 is applied, adding **reclen_t**.

Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

B.2.11.2 The char Type

POSIX.1-2024 explicitly requires that a **char** type is exactly one byte (8 bits).

B.2.12 Status Information

POSIX.1-2024 does not require all matching WNOWAIT threads (threads in a matching call to *waitid*( ) with the WNOWAIT flag set) to obtain a child's status information because the status information might be discarded (consumed or replaced) before one of the matching WNOWAIT threads is scheduled. If the status information is not discarded, it will remain available, so all of the matching WNOWAIT threads will (eventually) obtain the status information.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0031 [690] is applied.

B.3 System Interfaces

See the RATIONALE sections on the individual reference pages.

B.3.1 System Interfaces Removed in this Version

This section contains a list of options and interfaces removed in POSIX.1-2024, together with advice for application developers on the alternative interfaces that should be used.

B.3.1.1 STREAMS Option

Applications are recommended to use UNIX domain sockets as an alternative for much of the functionality provided by this option. For example, file descriptor passing can be performed using *sendmsg*( ) and *recvmsg*( ) with SCM_RIGHTS on a UNIX domain socket instead of using *ioctl*( ) with I_SENDFD and I_RECVFD on a STREAM.

B.3.1.2 Tracing Option

Applications are recommended to use implementation-provided extension interfaces instead of the functionality provided by this option. (Such interfaces were in widespread use before the Tracing option was added to POSIX.1 and continued to be used in preference to the Tracing option interfaces.)

B.3.1.3 _longjmp( ) and _setjmp( )

Applications are recommended to use *siglongjmp*( ) and *sigsetjmp*( ) instead of these functions.

B.3.1.4 _tolower( ) and _toupper( )

Applications are recommended to use *tolower*( ) and *toupper*( ) instead of these functions.

B.3.1.5 ftw( )

Applications are recommended to use *nftw*( ) instead of this function.

132945 ### *B.3.1.6 getitimer( ) and setitimer( )*

132946 Applications are recommended to use *timer_gettime*( ) and *timer_settime*( ) instead of these
132947 functions.

132948 ### *B.3.1.7 gets( )*

132949 Applications are recommended to use *fgets*( ) instead of this function.

132950 ### *B.3.1.8 gettimeofday( )*

132951 Applications are recommended to use *clock_gettime*( ) instead of this function.

132952 ### *B.3.1.9 isascii( ) and toascii( )*

132953 Applications are recommended to use macros equivalent to the following instead of these
132954 functions:

132955 ```
#define isascii(c) (((c) & ~0177) == 0)
```
132956 ```
#define toascii(c) ((c) & 0177)
```

132957 An alternative replacement for *isascii*( ), depending on the intended outcome if the code is
132958 ported to implementations with different character encodings, might be:

132959 ```
#define isascii(c) (isprint((c)) || iscntrl((c)))
```

132960 (In the C or POSIX locale, this determines whether *c* is a character in the portable character set.)

132961 ### *B.3.1.10 pthread_getconcurrency( ) and pthread_setconcurrency( )*

132962 Applications are recommended to use thread scheduling (on implementations that support the
132963 Thread Execution Scheduling option) instead of these functions; see XSH Section 2.9.4 (on page
132964 540).

132965 ### *B.3.1.11 rand_r( )*

132966 Applications are recommended to use *nrand48*( ) or *random*( ) instead of this function.

132967 ### *B.3.1.12 setpgrp( )*

132968 Applications are recommended to use *setpgid*( ) or *setsid*( ) instead of this function.

132969 ### *B.3.1.13 sighold( ), sigpause( ), and sigrelse( )*

132970 Applications are recommended to use *pthread_sigmask*( ) or *sigprocmask*( ) instead of these
132971 functions.

132972 ### *B.3.1.14 sigignore( ), siginterrupt( ), and sigset( )*

132973 Applications are recommended to use *sigaction*( ) instead of these functions.

  
```

132974 *B.3.1.15 tempnam( )*

132975 Applications are recommended to use *mkdtemp*( ), *mkstemp*( ), or *tmpfile*( ) instead of this
132976 function.

132977 *B.3.1.16 ulimit( )*

132978 Applications are recommended to use *getrlimit*( ) or *setrlimit*( ) instead of this function.

132979 *B.3.1.17 utime( )*

132980 Applications are recommended to use *futimens*( ) if a file descriptor for the file is open, otherwise
132981 *utimensat*( ), instead of this function.

132982 B.3.2 System Interfaces Removed in the Previous Version

132983 The functions and symbols removed in Issue 7 (from the Issue 6 base document) were as
132984 follows:

132985

Removed Functions and Symbols in Issue 7		
132986 *bcmp*( )	*gethostbyaddr*( )	*rindex*( )
132987 *bcopy*( )	*gethostbyname*( )	*scalb*( )
132988 *bsd_signal*( )	*getwd*( )	*setcontext*( )
132989 *bzero*( )	*h_errno*	*swapcontext*( )
132990 *ecvt*( )	*index*( )	*ualarm*( )
132991 *fcvt*( )	*makecontext*( )	*usleep*( )
132992 *ftime*( )	*mktemp*( )	*vfork*( )
132993 *gcvt*( )	*pthread_attr_getstackaddr*( )	*wcswcs*( )
132994 *getcontext*( )	*pthread_attr_setstackaddr*( )	

132995 B.3.3 Examples for Spawn

132996 The following long examples are provided in the Rationale (Informative) volume of
132997 POSIX.1-2024 as a supplement to the reference page for *posix_spawn*( ).

132998 **Example Library Implementation of Spawn**

132999 The *posix_spawn*( ) or *posix_spawnp*( ) functions provide the following:

133000 • Simply start a process executing a process image. This is the simplest application for
133001 process creation, and it may cover most executions of *fork*( ).

133002 • Support I/O redirection, including pipes.

133003 • Run the child under a user and group ID in the domain of the parent.

133004 • Run the child at any priority in the domain of the parent.

133005 The *posix_spawn*( ) or *posix_spawnp*( ) functions do not cover every possible use of the *fork*( )
133006 function, but they do span the common applications: typical use by a shell and a login utility.

133007 The price for an application is that before it calls *posix_spawn*( ) or *posix_spawnp*( ), the parent
133008 must adjust to a state that *posix_spawn*( ) or *posix_spawnp*( ) can map to the desired state for the
133009 child. Environment changes require the parent to save some of its state and restore it afterwards.
133010 The example below demonstrates an initial approach to implementing *posix_spawn*( ) using other

POSIX operations, although an actual implementation will need to be more robust at handling all possible filenames.

```
#include <sys/types.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sched.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>
#include <string.h>
#include <signal.h>

/* #include <spawn.h> */
/*****************************************/
/* Things that could be defined in spawn.h */
/*****************************************/
typedef struct
{
    short posix_attr_flags;
#define POSIX_SPAWN_SETPGROUP       0x1
#define POSIX_SPAWN_SETSIGMASK      0x2
#define POSIX_SPAWN_SETSIGDEF       0x4
#define POSIX_SPAWN_SETSCHEDULER    0x8
#define POSIX_SPAWN_SETSCHEDPARAM   0x10
#define POSIX_SPAWN_RESETIDS        0x20
#define POSIX_SPAWN_SETSID          0x40
    pid_t posix_attr_pgroup;
    sigset_t posix_attr_sigmask;
    sigset_t posix_attr_sigdefault;
    int posix_attr_schedpolicy;
    struct sched_param posix_attr_schedparam;
}   posix_spawnattr_t;

typedef char *posix_spawn_file_actions_t;

int posix_spawn_file_actions_init(
    posix_spawn_file_actions_t *file_actions);
int posix_spawn_file_actions_destroy(
    posix_spawn_file_actions_t *file_actions);
int posix_spawn_file_actions_addchdir(
    posix_spawn_file_actions_t *restrict file_actions,
    const char *restrict path);
int posix_spawn_file_actions_addclose(
    posix_spawn_file_actions_t *file_actions, int fildes);
int posix_spawn_file_actions_adddup2(
    posix_spawn_file_actions_t *file_actions, int fildes,
    int newfildes);
int posix_spawn_file_actions_addfchdir(
    posix_spawn_file_actions_t *file_actions, int fildes);
int posix_spawn_file_actions_addopen(
    posix_spawn_file_actions_t *file_actions, int fildes,
    const char *path, int oflag, mode_t mode);
int posix_spawnattr_init(posix_spawnattr_t *attr);
```



```
133062        int posix_spawnattr_destroy(posix_spawnattr_t *attr);
133063        int posix_spawnattr_getflags(const posix_spawnattr_t *attr,
133064            short *lags);
133065        int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
133066        int posix_spawnattr_getpgroup(const posix_spawnattr_t *attr,
133067            pid_t *pgroup);
133068        int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
133069        int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *attr,
133070            int *schedpolicy);
133071        int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
133072            int schedpolicy);
133073        int posix_spawnattr_getschedparam(const posix_spawnattr_t *attr,
133074            struct sched_param *schedparam);
133075        int posix_spawnattr_setschedparam(posix_spawnattr_t *attr,
133076            const struct sched_param *schedparam);
133077        int posix_spawnattr_getsigmask(const posix_spawnattr_t *attr,
133078            sigset_t *sigmask);
133079        int posix_spawnattr_setsigmask(posix_spawnattr_t *attr,
133080            const sigset_t *sigmask);
133081        int posix_spawnattr_getdefault(const posix_spawnattr_t *attr,
133082            sigset_t *sigdefault);
133083        int posix_spawnattr_setsigdefault(posix_spawnattr_t *attr,
133084            const sigset_t *sigdefault);
133085        int posix_spawn(pid_t *pid, const char *path,
133086            const posix_spawn_file_actions_t *file_actions,
133087            const posix_spawnattr_t *attrp, char *const argv[],
133088            char *const envp[]);
133089        int posix_spawnp(pid_t *pid, const char *file,
133090            const posix_spawn_file_actions_t *file_actions,
133091            const posix_spawnattr_t *attrp, char *const argv[],
133092            char *const envp[]);
133093        /****************************************/
133094        /* Example posix_spawn() library routine */
133095        /****************************************/
133096        int posix_spawn(pid_t *pid,
133097            const char *path,
133098            const posix_spawn_file_actions_t *file_actions,
133099            const posix_spawnattr_t *attrp,
133100            char *const argv[],
133101            char *const envp[])
133102        {
133103            /* Create process */
133104            if ((*pid = fork()) == (pid_t) 0)
133105            {
133106                /* This is the child process */
133107                /* Handle creating a new session */
133108                if (attrp->posix_attr_flags & POSIX_SPAWN_SETSID)
133109                {
133110                    /* Create a new session */
133111                    if (setsid() == -1)
133112                    {
133113                        /* Failed */
```



```
                        _exit(127);
                    }
                }

                /* Handle process group */
                if (attrp->posix_attr_flags & POSIX_SPAWN_SETPGROUP)
                {
                    /* Override inherited process group */
                    if (setpgid(0, attrp->posix_attr_pgroup) != 0)
                    {
                        /* Failed */
                        _exit(127);
                    }
                }

                /* Handle thread signal mask */
                if (attrp->posix_attr_flags & POSIX_SPAWN_SETSIGMASK)
                {
                    /* Set the signal mask (cannot fail) */
                    sigprocmask(SIG_SETMASK, &attrp->posix_attr_sigmask, NULL);
                }

                /* Handle resetting effective user and group IDs */
                if (attrp->posix_attr_flags & POSIX_SPAWN_RESETIDS)
                {
                    /* None of these can fail for this case. */
                    setuid(getuid());
                    setgid(getgid());
                }

                /* Handle defaulted signals */
                if (attrp->posix_attr_flags & POSIX_SPAWN_SETSIGDEF)
                {
                    struct sigaction deflt;
                    sigset_t all_signals;

                    int s;

                    /* Construct default signal action */
                    deflt.sa_handler = SIG_DFL;
                    deflt.sa_flags = 0;

                    /* Construct the set of all signals */
                    sigfillset(&all_signals);

                    /* Loop for all signals */
                    for (s = 0; sigismember(&all_signals, s); s++)
                    {
                        /* Signal to be defaulted? */
                        if (sigismember(&attrp->posix_attr_sigdefault, s))
                        {
                            /* Yes; default this signal */
                            if (sigaction(s, &deflt, NULL) == -1)
                            {
                                /* Failed */
                                _exit(127);
                            }
```



```
                }
            }
        }

        /* Handle the fds if they are to be mapped */
        if (file_actions != NULL)
        {
            /* Loop for all actions in object file_actions */
            /* (implementation dives beneath abstraction) */
            char *p = *file_actions;

            while (*p != '\0')
            {
                if (strncmp(p, "close(", 6) == 0)
                {
                    int fd;

                    if (sscanf(p + 6, "%d)", &fd) != 1)
                    {
                        _exit(127);
                    }
                    if (close(fd) == -1 && errno != EBADF)
                        _exit(127);
                }
                else if (strncmp(p, "dup2(", 5) == 0)
                {
                    int fd, newfd;

                    if (sscanf(p + 5, "%d,%d)", &fd, &newfd) != 2)
                    {
                        _exit(127);
                    }
                    if (fd == newfd)
                    {
                        int flags = fcntl(fd, F_GETFD);
                        if (flags == -1)
                            _exit(127);
                        flags &= ~FD_CLOEXEC;
                        if (fcntl(fd, F_SETFD, flags) == -1)
                            _exit(127);
                    }
                    else if (dup2(fd, newfd) == -1)
                        _exit(127);
                }
                else if (strncmp(p, "open(", 5) == 0)
                {
                    int fd, oflag;
                    mode_t mode;
                    int tempfd;
                    char path[1000];    /* Should be dynamic */
                    char *q;

                    if (sscanf(p + 5, "%d,", &fd) != 1)
                    {
                        _exit(127);
```



```
133213                  }
133214                  p = strchr(p, ',') + 1;
133215                  q = strchr(p, '*');
133216                  if (q == NULL)
133217                      _exit(127);
133218                  strncpy(path, p, q - p);
133219                  path[q - p] = '\0';
133220                  if (sscanf(q + 1, "%o,%o)", &oflag, &mode) != 2)
133221                  {
133222                      _exit(127);
133223                  }
133224                  if (close(fd) == -1)
133225                  {
133226                      if (errno != EBADF)
133227                          _exit(127);
133228                  }
133229                  tempfd = open(path, oflag, mode);
133230                  if (tempfd == -1)
133231                      _exit(127);
133232                  if (tempfd != fd)
133233                  {
133234                      if (dup2(tempfd, fd) == -1)
133235                      {
133236                          _exit(127);
133237                      }
133238                      if (close(tempfd) == -1)
133239                      {
133240                          _exit(127);
133241                      }
133242                  }
133243              }
133244              else if (strncmp(p, "chdir(", 6) == 0)
133245              {
133246                  char path[1000]; /* Should be dynamic */
133247                  char *q;

133248                  p += 6
133249                  q = strchr(p, '*');
133250                  if (q == NULL)
133251                      _exit(127);
133252                  strncpy(path, p, q - p);
133253                  path[q - p] = '\0';
133254                  if (chdir(path) == -1)
133255                      _exit(127);
133256              }
133257              else if (strncmp(p, "fchdir(", 7) == 0)
133258              {
133259                  int fd;

133260                  if (sscanf(p + 7, "%d)", &fd) != 1)
133261                      _exit(127);
133262                  if (fchdir(fd) == -1)
133263                      _exit(127);
```



```
                        }
                        else
                        {
                            _exit(127);
                        }
                        p = strchr(p, ')') + 1;
                    }
                }

                /* Handle setting new scheduling policy and parameters */
                if (attrp->posix_attr_flags & POSIX_SPAWN_SETSCHEDULER)
                {
                    if (sched_setscheduler(0, attrp->posix_attr_schedpolicy,
                        &attrp->posix_attr_schedparam) == -1)
                    {
                        _exit(127);
                    }
                }

                /* Handle setting only new scheduling parameters */
                if (attrp->posix_attr_flags & POSIX_SPAWN_SETSCHEDPARAM)
                {
                    if (sched_setparam(0, &attrp->posix_attr_schedparam) == -1)
                    {
                        _exit(127);
                    }
                }

                /* Now execute the program at path */
                /* Any fd that still has FD_CLOEXEC set will be closed */
                execve(path, argv, envp);
                _exit(127);             /* exec failed */
            }
            else
            {
                /* This is the parent (calling) process */
                if (*pid == (pid_t) - 1)
                    return errno;
                return 0;
            }
        }

        /*******************************************************/
        /* Here is a crude but effective implementation of the */
        /* file action object operators which store actions as */
        /* concatenated token-separated strings.               */
        /*******************************************************/
        /* Create object with no actions. */
        int posix_spawn_file_actions_init(
            posix_spawn_file_actions_t *file_actions)
        {
            *file_actions = malloc(sizeof(char));
            if (*file_actions == NULL)
                return ENOMEM;
            strcpy(*file_actions, "");
```



```
133315          return 0;
133316      }
133317      /* Free object storage and make invalid. */
133318      int posix_spawn_file_actions_destroy(
133319          posix_spawn_file_actions_t *file_actions)
133320      {
133321          free(*file_actions);
133322          *file_actions = NULL;
133323          return 0;
133324      }
133325      /* Add a new action string to object. */
133326      static int add_to_file_actions(
133327          posix_spawn_file_actions_t *file_actions, char *new_action)
133328      {
133329          *file_actions = realloc
133330          (*file_actions, strlen(*file_actions) + strlen(new_action) + 1);
133331          if (*file_actions == NULL)
133332              return ENOMEM;
133333          strcat(*file_actions, new_action);
133334          return 0;
133335      }
133336      /* Add a chdir action to object. */
133337      int posix_spawn_file_actions_addchdir(
133338          posix_spawn_file_actions_t *restrict file_actions,
133339          const char *restrict path)
133340      {
133341          char temp[100];
133342          sprintf(temp, "chdir(%s*)", path);
133343          return add_to_file_actions(file_actions, temp);
133344      }
133345      /* Add a close action to object. */
133346      int posix_spawn_file_actions_addclose(
133347          posix_spawn_file_actions_t *file_actions, int fildes)
133348      {
133349          char temp[100];
133350          sprintf(temp, "close(%d)", fildes);
133351          return add_to_file_actions(file_actions, temp);
133352      }
133353      /* Add a dup2 action to object. */
133354      int posix_spawn_file_actions_adddup2(
133355          posix_spawn_file_actions_t *file_actions, int fildes,
133356          int newfildes)
133357      {
133358          char temp[100];
133359          sprintf(temp, "dup2(%d,%d)", fildes, newfildes);
133360          return add_to_file_actions(file_actions, temp);
133361      }
133362      /* Add a fchdir action to object. */
```



```
133363        int posix_spawn_file_actions_addfchdir(
133364            posix_spawn_file_actions_t *file_actions, int fildes)
133365        {
133366            char temp[100];
133367            sprintf(temp, "fchdir(%d)", fildes);
133368            return add_to_file_actions(file_actions, temp);
133369        }
133370        /* Add an open action to object. */
133371        int posix_spawn_file_actions_addopen(
133372            posix_spawn_file_actions_t *file_actions, int fildes,
133373            const char *path, int oflag, mode_t mode)
133374        {
133375            char temp[100];
133376            sprintf(temp, "open(%d,%s*%o,%o)", fildes, path, oflag, mode);
133377            return add_to_file_actions(file_actions, temp);
133378        }
133379        /*******************************************************/
133380        /* Here is a crude but effective implementation of the */
133381        /* spawn attributes object functions which manipulate  */
133382        /* the individual attributes.                          */
133383        /*******************************************************/
133384        /* Initialize object with default values. */
133385        int posix_spawnattr_init(posix_spawnattr_t *attr)
133386        {
133387            attr->posix_attr_flags = 0;
133388            attr->posix_attr_pgroup = 0;
133389            /* Default value of signal mask is the parent's signal mask; */
133390            /* other values are also allowed */
133391            sigprocmask(0, NULL, &attr->posix_attr_sigmask);
133392            sigemptyset(&attr->posix_attr_sigdefault);
133393            /* Default values of scheduling attr inherited from the parent; */
133394            /* other values are also allowed */
133395            attr->posix_attr_schedpolicy = sched_getscheduler(0);
133396            sched_getparam(0, &attr->posix_attr_schedparam);
133397            return 0;
133398        }
133399        int posix_spawnattr_destroy(posix_spawnattr_t *attr)
133400        {
133401            /* No action needed */
133402            return 0;
133403        }
133404        int posix_spawnattr_getflags(const posix_spawnattr_t *attr,
133405            short *flags)
133406        {
133407            *flags = attr->posix_attr_flags;
133408            return 0;
133409        }
133410        int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags)
133411        {
```



```
133412          attr->posix_attr_flags = flags;
133413          return 0;
133414      }
133415      int posix_spawnattr_getpgroup(const posix_spawnattr_t *attr,
133416          pid_t *pgroup)
133417      {
133418          *pgroup = attr->posix_attr_pgroup;
133419          return 0;
133420      }
133421      int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup)
133422      {
133423          attr->posix_attr_pgroup = pgroup;
133424          return 0;
133425      }
133426      int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *attr,
133427          int *schedpolicy)
133428      {
133429          *schedpolicy = attr->posix_attr_schedpolicy;
133430          return 0;
133431      }
133432      int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
133433          int schedpolicy)
133434      {
133435          attr->posix_attr_schedpolicy = schedpolicy;
133436          return 0;
133437      }
133438      int posix_spawnattr_getschedparam(const posix_spawnattr_t *attr,
133439          struct sched_param *schedparam)
133440      {
133441          *schedparam = attr->posix_attr_schedparam;
133442          return 0;
133443      }
133444      int posix_spawnattr_setschedparam(posix_spawnattr_t *attr,
133445          const struct sched_param *schedparam)
133446      {
133447          attr->posix_attr_schedparam = *schedparam;
133448          return 0;
133449      }
133450      int posix_spawnattr_getsigmask(const posix_spawnattr_t *attr,
133451          sigset_t *sigmask)
133452      {
133453          *sigmask = attr->posix_attr_sigmask;
133454          return 0;
133455      }
133456      int posix_spawnattr_setsigmask(posix_spawnattr_t *attr,
133457          const sigset_t *sigmask)
133458      {
133459          attr->posix_attr_sigmask = *sigmask;
133460          return 0;
```



```
}

int posix_spawnattr_getsigdefault(const posix_spawnattr_t *attr,
    sigset_t *sigdefault)
{
    *sigdefault = attr->posix_attr_sigdefault;
    return 0;
}

int posix_spawnattr_setsigdefault(posix_spawnattr_t *attr,
    const sigset_t *sigdefault)
{
    attr->posix_attr_sigdefault = *sigdefault;
    return 0;
}
```

I/O Redirection with Spawn

I/O redirection with *posix_spawn*() or *posix_spawnp*() is accomplished by crafting a *file_actions* argument to effect the desired redirection. Such a redirection follows the general outline of the following example:

```
/* To redirect new standard output (fd 1) to a file, */
/* and redirect new standard input (fd 0) from my fd socket_pair[1], */
/* and close my fd socket_pair[0] in the new process. */
posix_spawn_file_actions_t file_actions;
posix_spawn_file_actions_init(&file_actions);
posix_spawn_file_actions_addopen(&file_actions, 1, "newout", ...);
posix_spawn_file_actions_dup2(&file_actions, socket_pair[1], 0);
posix_spawn_file_actions_close(&file_actions, socket_pair[0]);
posix_spawn_file_actions_close(&file_actions, socket_pair[1]);
posix_spawn(..., &file_actions, ...);
posix_spawn_file_actions_destroy(&file_actions);
```

Spawning a Process Under a New User ID

Spawning a process under a new user ID follows the outline shown in the following example:

```
Save = getuid();
setuid(newid);
posix_spawn(...);
setuid(Save);
```


Rationale (Informative)

Part C:

Shell and Utilities

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

133500 *Appendix C*

133501 # *Rationale for Shell and Utilities*

133502 ## C.1 Introduction

133503 ### C.1.1 Change History

133504 The change history is provided as an informative section, to track changes from earlier versions
133505 of this standard.

133506 The following sections describe changes made to the Shell and Utilities volume of POSIX.1-2024
133507 since Issue 7 of the base document. The CHANGE HISTORY section for each utility describes
133508 technical changes made to that utility in Issue 5 and later. Changes made before Issue 5 are not
133509 included.

133510 **Changes from Issue 7 to Issue 8 (POSIX.1-2024)**

133511 The following list summarizes the major changes that were made in the Shell and Utilities
133512 volume of POSIX.1-2024 from Issue 7 to Issue 8:

133513 • The Open Group Standard, 2022, Additional APIs for the Base Specifications Issue 8, Part 2
133514 is incorporated.

133515 • Austin Group defect reports and IEEE Interpretations against IEEE Std 1003.1 are applied.

133516 • The Open Group corrigenda and resolutions are applied.

133517 • Features, marked obsolescent in the base document, have been considered for removal in
133518 this version.

133519 **New Features in Issue 8**

133520 The utilities first introduced in Issue 8 (over the Issue 7 base document) are as follows:

133521

New Utilities in Issue 8		
133522 *gettext*	*readlink*	*xgettext*
133523 *msgfmt*	*realpath*	
133524 *ngettext*	*timeout*	

133525 **Removed Utilities in Issue 8**

133526 The utilities removed in Issue 8 (from the Issue 7 base document) are as follows:

133527

Removed Utilities in Issue 8		
133528 *fort77*	*qmove*	*qselect*
133529 *qalter*	*qmsg*	*qsig*
133530 *qdel*	*qrerun*	*qstat*
133531 *qhold*	*qrls*	*qsub*

C.1.2 Relationship to Other Documents

C.1.2.1 System Interfaces

It has been pointed out that the Shell and Utilities volume of POSIX.1-2024 assumes that a great deal of functionality from the System Interfaces volume of POSIX.1-2024 is present, but never states exactly how much (and strictly does not need to since both are mandated on a conforming system). This section is an attempt to clarify the assumptions.

File Read, Write, and Creation

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/2 is applied, updating Table 1-1.

File Removal

This is intended to be a summary of the *unlink*() and *rmdir*() requirements. Note that it is possible using the *unlink*() function for item 4. to occur.

C.1.2.2 Concepts Derived from the ISO C Standard

This section was introduced to address the issue that there was insufficient detail presented by such utilities as *awk* or *sh* about their procedural control statements and their methods of performing arithmetic functions.

The ISO C standard was selected as a model because most historical implementations of the standard utilities were written in C. Thus, it was more likely that they would act in the desired manner without modification.

Using the ISO C standard is primarily a notational convenience so that the many procedural languages in the Shell and Utilities volume of POSIX.1-2024 would not have to be rigorously described in every aspect. Its selection does not require that the standard utilities be written in Standard C; they could be written in Common Usage C, Ada, Pascal, assembler language, or anything else.

The sizes of the various numeric values refer to C-language data types that are allowed to be different sizes by the ISO C standard. Thus, like a C-language application, a shell application cannot rely on their exact size. However, it can rely on their minimum sizes expressed in the ISO C standard, such as {LONG_MAX} for a **long** type.

The behavior on overflow is undefined for ISO C standard arithmetic. Therefore, the standard utilities can use ``bignum'' representation for integers so that there is no fixed maximum unless otherwise stated in the utility description. Similarly, standard utilities can use infinite-precision representations for floating-point arithmetic, as long as these representations exceed the ISO C standard requirements.

This section addresses only the issue of semantics; it is not intended to specify syntax. For example, the ISO C standard requires that 0L be recognized as an integer constant equal to zero, but utilities such as *awk* and *sh* are not required to recognize 0L (though they are allowed to, as an extension).

The ISO C standard requires that a C compiler must issue a diagnostic for constants that are too large to represent. Most standard utilities are not required to issue these diagnostics; for example, the command:

```
diff -C 2147483648 file1 file2
```


133572 has undefined behavior, and the *diff* utility is not required to issue a diagnostic even if the
133573 number 2 147 483 648 cannot be represented.

133574 Austin Group Defect 1128 is applied, adding a note about the comma operator.

133575 C.1.3 Utility Limits

133576 This section grew out of an idea that originated with the original POSIX.1, in the tables of system
133577 limits for the *sysconf*( ) and *pathconf*( ) functions. The idea being that a conforming application
133578 can be written to use the most restrictive values that a minimal system can provide, but it should
133579 not have to. The values provided represent compromises so that some vendors can use
133580 historically limited versions of UNIX system utilities. They are the highest values that a strictly
133581 conforming application can assume, given no other information.

133582 However, by using the *getconf* utility or the *sysconf*( ) function, the elegant application can be
133583 tailored to more liberal values on some of the specific instances of specific implementations.

133584 There is no explicitly stated requirement that an implementation provide finite limits for any of
133585 these numeric values; the implementation is free to provide essentially unbounded capabilities
133586 (where it makes sense), stopping only at reasonable points such as {ULONG_MAX} (from the
133587 ISO C standard). Therefore, applications desiring to tailor themselves to the values on a
133588 particular implementation need to be ready for possibly huge values; it may not be a good idea
133589 to allocate blindly a buffer for an input line based on the value of {LINE_MAX}, for instance.
133590 However, unlike the System Interfaces volume of POSIX.1-2024, there is no set of limits that
133591 return a special indication meaning ``unbounded''. The implementation should always return an
133592 actual number, even if the number is very large.

133593 The statement:

133594 > ``It is not guaranteed that the application ...''

133595 is an indication that many of these limits are designed to ensure that implementors design their
133596 utilities without arbitrary constraints related to unimaginative programming. There are certainly
133597 conditions under which combinations of options can cause failures that would not render an
133598 implementation non-conforming. For example, {EXPR_NEST_MAX} and {ARG_MAX} could
133599 collide when expressions are large; combinations of {BC_SCALE_MAX} and {BC_DIM_MAX}
133600 could exceed virtual memory.

133601 In the Shell and Utilities volume of POSIX.1-2024, the notion of a limit being guaranteed for the
133602 process lifetime, as it is in the System Interfaces volume of POSIX.1-2024, is not as useful to a
133603 shell script. The *getconf* utility is probably a process itself, so the guarantee would be without
133604 value. Therefore, the Shell and Utilities volume of POSIX.1-2024 requires the guarantee to be for
133605 the session lifetime. This will mean that many vendors will either return very conservative
133606 values or possibly implement *getconf* as a built-in.

133607 It may seem confusing to have limits that apply only to a single utility grouped into one global
133608 section. However, the alternative, which would be to disperse them out into their utility
133609 description sections, would cause great difficulty when *sysconf*( ) and *getconf* were described.
133610 Therefore, the standard developers chose the global approach.

133611 Each language binding could provide symbol names that are slightly different from those shown
133612 here. For example, the C-Language Binding option adds a leading <underscore> to the symbols
133613 as a prefix.

133614 The following comments describe selection criteria for the symbols and their values:

133615 {ARG_MAX}
133616 This is defined by the System Interfaces volume of POSIX.1-2024. Unfortunately, it is very
133617 difficult for a conforming application to deal with this value, as it does not know how much
133618 of its argument space is being consumed by the environment variables of the user.

133619 {BC_BASE_MAX}
133620 {BC_DIM_MAX}
133621 {BC_SCALE_MAX}
133622 These were originally one value, {BC_SCALE_MAX}, but it was unreasonable to link all
133623 three concepts into one limit.

133624 {CHILD_MAX}
133625 This is defined by the System Interfaces volume of POSIX.1-2024.

133626 {COLL_WEIGHTS_MAX}
133627 The weights assigned to **order** can be considered as ``passes'' through the collation
133628 algorithm.

133629 {EXPR_NEST_MAX}
133630 The value for expression nesting was borrowed from the ISO C standard.

133631 {LINE_MAX}
133632 This is a global limit that affects all utilities, unless otherwise noted. The {MAX_CANON}
133633 value from the System Interfaces volume of POSIX.1-2024 may further limit input lines from
133634 terminals. The {LINE_MAX} value was the subject of much debate and is a compromise
133635 between those who wished to have unlimited lines and those who understood that many
133636 historical utilities were written with fixed buffers. Frequently, utility writers selected the
133637 UNIX system constant BUFSIZ to allocate these buffers; therefore, some utilities were
133638 limited to 512 bytes for I/O lines, while others achieved 4 096 bytes or greater.

133639 It should be noted that {LINE_MAX} applies only to input line length; there is no
133640 requirement in POSIX.1-2024 that limits the length of output lines. Utilities such as *awk*, *sed*,
133641 and *paste* could theoretically construct lines longer than any of the input lines they received,
133642 depending on the options used or the instructions from the application. They are not
133643 required to truncate their output to {LINE_MAX}. It is the responsibility of the application
133644 to deal with this. If the output of one of those utilities is to be piped into another of the
133645 standard utilities, line length restrictions will have to be considered; the *fold* utility, among
133646 others, could be used to ensure that only reasonable line lengths reach utilities or
133647 applications.

133648 {LINK_MAX}
133649 This is defined by the System Interfaces volume of POSIX.1-2024.

133650 {MAX_CANON}
133651 {MAX_INPUT}
133652 {NAME_MAX}
133653 {NGROUPS_MAX}
133654 {OPEN_MAX}
133655 {PATH_MAX}
133656 {PIPE_BUF}
133657 These limits are defined by the System Interfaces volume of POSIX.1-2024. Note that the
133658 byte lengths described by some of these values continue to represent bytes, even if the
133659 applicable character set uses a multi-byte encoding.

133660 {RE_DUP_MAX}
133661 The value selected is consistent with historical practice. Although the name implies that it
133662 applies to all REs, only BREs use the interval notation \{*m*,*n*\} addressed by this limit.

133663 {POSIX2_SYMLINKS}

133664 The {POSIX2_SYMLINKS} variable indicates that the underlying operating system supports
133665 the creation of symbolic links in specific directories. Many of the utilities defined in
133666 POSIX.1-2024 that deal with symbolic links do not depend on this value. For example, a
133667 utility that follows symbolic links (or does not, as the case may be) will only be affected by a
133668 symbolic link if it encounters one. Presumably, a file system that does not support symbolic
133669 links will not contain any. This variable does affect such utilities as *ln* **–s** and *pax* that
133670 attempt to create symbolic links.

133671 There are different limits associated with command lines and input to utilities, depending on the
133672 method of invocation. In the case of a C program *exec*-ing a utility, {ARG_MAX} is the
133673 underlying limit. In the case of the shell reading a script and *exec*-ing a utility, {LINE_MAX}
133674 limits the length of lines the shell is required to process, and {ARG_MAX} will still be a limit. If a
133675 user is entering a command on a terminal to the shell, requesting that it invoke the utility,
133676 {MAX_INPUT} may restrict the length of the line that can be given to the shell to a value below
133677 {LINE_MAX}.

133678 When an option is supported, *getconf* returns a value of 1. For example, when C development is
133679 supported:

```
133680 if [ "$(getconf POSIX2_C_DEV)" -eq 1 ]; then
133681     echo C supported
133682 fi
```

133683 The *sysconf*( ) function in the C-Language Binding option would return 1.

133684 The following comments describe selection criteria for the symbols and their values:

133685 POSIX2_C_BIND
133686 POSIX2_C_DEV
133687 POSIX2_FORT_RUN
133688 POSIX2_SW_DEV
133689 POSIX2_UPE

133690 It is possible for some (usually privileged) operations to remove utilities that support these
133691 options or otherwise to render these options unsupported. The header files, the *sysconf*( )
133692 function, or the *getconf* utility will not necessarily detect such actions, in which case they
133693 should not be considered as rendering the implementation non-conforming. A test suite
133694 should not attempt tests such as:

```
133695     rm /usr/bin/c17
133696     getconf POSIX2_C_DEV
```

133697 POSIX2_LOCALEDEF

133698 This symbol was introduced to allow implementations to restrict supported locales to only
133699 those supplied by the implementation.

133700 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/2 is applied, deleting the entry for
133701 {POSIX2_VERSION} since it is not a utility limit minimum value.

133702 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/3 is applied, changing the text in Utility
133703 Limits from: ``utility (see *getconf*) through the *sysconf*( ) function defined in the System Interfaces
133704 volume of POSIX.1-2024. The literal names shown in Table 1-3 apply only to the *getconf* utility;
133705 the high-level language binding describes the exact form of each name to be used by the
133706 interfaces in that binding.'' to: ``utility (see *getconf*).''.

133707 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0001 [666] is applied.

C.1.4 Grammar Conventions

There is no additional rationale provided for this section.

C.1.5 Utility Description Defaults

This section is arranged with headings in the same order as all the utility descriptions. It is a collection of related and unrelated information concerning:

1. The default actions of utilities
2. The meanings of notations used in POSIX.1-2024 that are specific to individual utility sections

Although this material may seem out of place here, it is important that this information appear before any of the utilities to be described later.

NAME

There is no additional rationale provided for this section.

SYNOPSIS

There is no additional rationale provided for this section.

DESCRIPTION

Austin Group Defect 351 is applied, adding a requirement relating to declaration utilities.

OPTIONS

Although it has not always been possible, the standard developers tried to avoid repeating information to reduce the risk that duplicate explanations could each be modified differently.

The need to recognize −− is required because conforming applications need to shield their operands from any arbitrary options that the implementation may provide as an extension. For example, if the standard utility *foo* is listed as taking no options, and the application needed to give it a pathname with a leading <hyphen-minus>, it could safely do it as:

```
foo -- -myfile
```

and avoid any problems with **−m** used as an extension.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0002 [584] is applied.

OPERANDS

The usage of − is never shown in the SYNOPSIS. Similarly, the usage of −− is never shown.

The requirement for processing operands in command-line order is to avoid a ``WeirdNIX'' utility that might choose to sort the input files alphabetically, by size, or by directory order. Although this might be acceptable for some utilities, in general the programmer has a right to know exactly what order will be chosen.

Some of the standard utilities take multiple *file* operands and act as if they were processing the concatenation of those files. For example:

```
asa file1 file2
```

and:


```
cat file1 file2 | asa
```

have similar results when questions of file access, errors, and performance are ignored. Other utilities such as *grep* or *wc* have completely different results in these two cases. This latter type of utility is always identified in its DESCRIPTION or OPERANDS sections, whereas the former is not. Although it might be possible to create a general assertion about the former case, the following points must be addressed:

- Access times for the files might be different in the operand case *versus* the *cat* case.
- The utility may have error messages that are cognizant of the input filename, and this added value should not be suppressed. (As an example, *awk* sets a variable with the filename at each file boundary.)

STDIN

There is no additional rationale provided for this section.

INPUT FILES

A conforming application cannot assume the following three commands are equivalent:

```
tail -n +2 file
(sed -n 1q; cat) < file
cat file | (sed -n 1q; cat)
```

The second command is equivalent to the first only when the file is seekable. In the third command, if the file offset in the open file description were not unspecified, *sed* would have to be implemented so that it read from the pipe 1 byte at a time or it would have to employ some method to seek backwards on the pipe. Such functionality is not defined currently in POSIX.1 and does not exist on all historical systems. Other utilities, such as *head*, *read*, and *sh*, have similar properties, so the restriction is described globally in this section.

The definition of ``text file'' is strictly enforced for input to the standard utilities; very few of them list exceptions to the undefined results called for here. (Of course, ``undefined'' here does not mean that historical implementations necessarily have to change to start indicating error conditions. Conforming applications cannot rely on implementations succeeding or failing when non-text files are used.)

The utilities that allow line continuation are generally those that accept input languages, rather than pure data. It would be unusual for an input line of this type to exceed {LINE_MAX} bytes and unreasonable to require that the implementation allow unlimited accumulation of multiple lines, each of which could reach {LINE_MAX}. Thus, for a conforming application the total of all the continued lines in a set cannot exceed {LINE_MAX}.

The format description is intended to be sufficiently rigorous to allow other applications to generate these input files. However, since <blank> characters can legitimately be included in some of the fields described by the standard utilities, particularly in locales other than the POSIX locale, this intent is not always realized.

133781 **ENVIRONMENT VARIABLES**

133782 There is no additional rationale provided for this section.

133783 **ASYNCHRONOUS EVENTS**

133784 Because there is no language prohibiting it, a utility is permitted to catch a signal, perform some
133785 additional processing (such as deleting temporary files), restore the default signal action, and
133786 resignal itself.

133787 Austin Group Defects 1648 and 1772 are applied, clarifying the default behavior for signal
133788 handling.

133789 **STDOUT**

133790 The format description is intended to be sufficiently rigorous to allow post-processing of output
133791 by other programs, particularly by an *awk* or *lex* parser.

133792 **STDERR**

133793 This section does not describe error messages that refer to incorrect operation of the utility.
133794 Consider a utility that processes program source code as its input. This section is used to
133795 describe messages produced by a correctly operating utility that encounters an error in the
133796 program source code on which it is processing. However, a message indicating that the utility
133797 had insufficient memory in which to operate would not be described.

133798 Some utilities have traditionally produced warning messages without returning a non-zero exit
133799 status; these are specifically noted in their sections. Other utilities shall not write to standard
133800 error if they complete successfully, unless the implementation provides some sort of extension to
133801 increase the verbosity or debugging level.

133802 The format descriptions are intended to be sufficiently rigorous to allow post-processing of
133803 output by other programs.

133804 **OUTPUT FILES**

133805 The format description is intended to be sufficiently rigorous to allow post-processing of output
133806 by other programs, particularly by an *awk* or *lex* parser.

133807 Receipt of the SIGQUIT signal should generally cause termination (unless in some debugging
133808 mode) that would bypass any attempted recovery actions.

133809 **EXTENDED DESCRIPTION**

133810 There is no additional rationale provided for this section.

133811 **EXIT STATUS**

133812 Note the additional discussion of exit values in *Exit Status for Commands* in the *sh* utility. It
133813 describes requirements for returning exit values greater than 125.

133814 A utility may list zero as a successful return, 1 as a failure for a specific reason, and greater than
133815 1 as ``an error occurred''. In this case, unspecified conditions may cause a 2 or 3, or other value,
133816 to be returned. A strictly conforming application should be written so that it tests for successful
133817 exit status values (zero in this case), rather than relying upon the single specific error value listed
133818 in POSIX.1-2024. In that way, it will have maximum portability, even on implementations with
133819 extensions.

133820 The standard developers are aware that the general non-enumeration of errors makes it difficult

to write test suites that test the *incorrect* operation of utilities. There are some historical implementations that have expended effort to provide detailed status messages and a helpful environment to bypass or explain errors, such as prompting, retrying, or ignoring unimportant syntax errors; other implementations have not. Since there is no realistic way to mandate system behavior in cases of undefined application actions or system problems—in a manner acceptable to all cultures and environments—attention has been limited to the correct operation of utilities by the conforming application. Furthermore, the conforming application does not need detailed information concerning errors that it caused through incorrect usage or that it cannot correct.

Austin Group Defect 1492 is applied, adding the **Default Behavior** paragraph.

CONSEQUENCES OF ERRORS

Several actions are possible when a utility encounters an error condition, depending on the severity of the error and the state of the utility. Included in the possible actions of various utilities are: deletion of temporary or intermediate work files; deletion of incomplete files; and validity checking of the file system or directory.

The text about recursive traversing is meant to ensure that utilities such as *find* process as many files in the hierarchy as they can. They should not abandon all of the hierarchy at the first error and resume with the next command-line operand, but should attempt to keep going.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0001 [150] is applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0003 [913] is applied.

Austin Group Defect 251 is applied, adding a note about the treatment of pathnames containing any bytes that have the encoded value of a <newline> character.

Austin Group Defect 1499 is applied, requiring utilities to exit with an exit status that indicates an error occurred, instead of any non-zero exit status.

APPLICATION USAGE

This section provides additional caveats, issues, and recommendations to the developer.

EXAMPLES

This section provides sample usage.

RATIONALE

There is no additional rationale provided for this section.

FUTURE DIRECTIONS

FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in the future, and often cautions the developer to architect the code to account for a change in this area. Note that a future directions statement should not be taken as a commitment to adopt a feature or interface in the future.

133855 **SEE ALSO**

133856 There is no additional rationale provided for this section.

133857 **CHANGE HISTORY**

133858 There is no additional rationale provided for this section.

133859 C.1.6 Considerations for Utilities in Support of Files of Arbitrary Size

133860 This section is intended to clarify the requirements for utilities in support of large files.

133861 The utilities listed in this section are utilities which are used to perform administrative tasks
133862 such as to create, move, copy, remove, change the permissions, or measure the resources of a file.
133863 They are useful both as end-user tools and as utilities invoked by applications during software
133864 installation and operation.

133865 The *chgrp*, *chmod*, *chown*, *ln*, and *rm* utilities probably require use of large file-capable versions of
133866 *stat*( ), *lstat*( ), *nftw*( ), and the **stat** structure.

133867 The *cat*, *cksum*, *cmp*, *cp*, *dd*, *mv*, and *touch* utilities probably require use of large file-capable
133868 versions of *creat*( ), *open*( ), and *fopen*( ).

133869 The *cat*, *cksum*, *cmp*, *dd*, *df*, *du*, and *ls* utilities may require writing large integer values. For
133870 example:

133871 • The *cat* utility might have a **−n** option which counts <newline> characters.

133872 • The *cksum* and *ls* utilities report file sizes.

133873 • The *cmp* utility reports the line number at which the first difference occurs, and also has a
133874 **−l** option which reports file offsets.

133875 • The *dd*, *df*, *du*, and *ls* utilities report block counts.

133876 The *dd*, *find*, and *test* utilities may need to interpret command arguments that contain 64-bit
133877 values. For *dd*, the arguments include *skip=n*, *seek=n*, and *count=n*. For *find*, the arguments
133878 include **−size***n*. For *test*, the arguments are those associated with algebraic comparisons.

133879 The *df* utility might need to access large file systems with *statvfs*( ).

133880 The *ulimit* utility will need to use large file-capable versions of *getrlimit*( ) and *setrlimit*( ) and be
133881 able to read and write large integer values.

133882 Austin Group Defect 1568 is applied, removing references to the *sum* utility.

133883 C.1.7 Built-In Utilities

133884 Other than the special built-in utilities, there is no requirement to build utilities into the shell
133885 itself. However, many shells implement certain utilities as regular built-ins for the following
133886 reasons:

133887 • To improve performance, especially for frequently used lightweight utilities (such as *test*,
133888 *true*, and *false*).

133889 • To eliminate the need for some sort of interprocess communication between the shell and
133890 those utilities that read or modify the shell's execution environment (such as *cd*).

- To make it easier to satisfy the command search and execution requirements in XCU Section 2.9.1.4 (on page 2502) for intrinsic utilities. Intrinsic utilities must be found prior to the *PATH* search. The shell could satisfy this requirement by keeping a list of the intrinsic utility pathnames and directly accessing the file-system versions regardless of *PATH*, but these utilities usually need to read or modify the shell's execution environment anyway.

With the exception of the intrinsic utilities, all regular built-in utilities are subject to the *PATH* search and can be overridden by a specially crafted *PATH* environment variable.

Earlier versions of this standard required that all of the regular built-in utilities, including intrinsic utilities, could be *exec*-ed. This was always a contentious requirement, and with the introduction of intrinsic utilities the standard developers decided to exempt the utilities that this standard requires to be intrinsic, with the exception of *kill*. The *kill* utility is still genuinely useful when *exec*-ed, only lacking support for the `%` job ID notation, whereas examples given of uses for the other utilities that are now exempted were considered contrived (such as using *cd* to test accessibility of a directory, which can be done using *test* **−x**). If an application needs *exec*-able versions of some of the exempted intrinsic utilities, it can easily provide them itself, on systems that support the (non-standard but ubiquitous) `"#!"` mechanism to make scripts executable by the *exec* family of functions, as links to a two-line shell script:

```
#! /path/to/sh
${0##*/} "$@"
```

Austin Group Defect 854 is applied, replacing the table of Regular Built-In Utilities with a reference to the new *Intrinsic Utilities* section.

Austin Group Defect 1600 is applied, exempting the intrinsic utilities other than *kill* from the requirement that they can be *exec*-ed.

C.1.8 Intrinsic Utilities

There were varying reasons for including utilities in the table of intrinsic utilities:

alias, *fc*, *unalias*
: The functionality of these utilities is performed more simply within the shell itself and that is the model most historical implementations have used.

bg, *fg*, *jobs*
: All of the job control-related utilities are eligible for built-in status because that is the model most historical implementations have used.

cd, *getopts*, *hash*, *read*, *type*, *ulimit*, *umask*, *wait*
: The functionality of these utilities is performed more simply within the context of the current process. An example can be taken from the usage of the *cd* utility. The purpose of the *cd* utility is to change the working directory for subsequent operations. The actions of *cd* affect the process in which *cd* is executed and all subsequent child processes of that process. Based on the POSIX standard process model, changes in the process environment of a child process have no effect on the parent process. If the *cd* utility were executed from a child process, the working directory change would be effective only in the child process. Child processes initiated subsequent to the child process that executed the *cd* utility would not have a changed working directory relative to the parent process.

command
: This utility was placed in the table primarily to protect scripts that are concerned about their *PATH* being manipulated. The ``secure'' shell script example in the *command* utility in the Shell and Utilities volume of POSIX.1-2024 would not be possible if a *PATH* change retrieved an alien version of *command*. (An alternative would have been to implement

getconf as a built-in, but the standard developers considered that it carried too many changing configuration strings to require in the shell.)

kill Since *kill* provides optional job control functionality using shell notation (`%1`, `%2`, and so on), some implementations would find it extremely difficult to provide this outside the shell.

The following utilities are frequently implemented as intrinsic (and built-in) utilities. Future versions of this standard might not allow these utilities, or any other standard utility not in Table 1-5 (on page 2470), to be intrinsic; implementations are encouraged to implement these as non-intrinsic utilities instead (but still built-in if they were previously built-in).

[, *echo*, *false*, *newgrp*, *printf*, *pwd*, *test*, *true*

All utilities, including those in the table, are accessible via the *system*() and *popen*() functions in the System Interfaces volume of POSIX.1-2024. There are situations where the return functionality of *system*() and *popen*() is not desirable. Applications that require the exit status of the invoked utility will not be able to use *system*() or *popen*(), since the exit status returned is that of the command language interpreter rather than that of the invoked utility. The alternative for such applications is the use of the *exec* family.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0004 [705] is applied.

Austin Group Defect 854 is applied, adding intrinsic utilities.

C.2 Shell Command Language

C.2.1 Shell Introduction

The System V shell was selected as the starting point for the Shell and Utilities volume of POSIX.1-2024. The BSD C shell was excluded from consideration for the following reasons:

- Most historically portable shell scripts assume the Version 7 Bourne shell, from which the System V shell is derived.
- The majority of tutorial materials on shell programming assume the System V shell.

The construct `"#!"` is reserved for implementations wishing to provide that extension. If it were not reserved, the Shell and Utilities volume of POSIX.1-2024 would disallow it by forcing it to be a comment. As it stands, a strictly conforming application must not use `"#!"` as the first two characters of the file.

Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

Austin Group Defect 1514 is applied, correcting a misuse of the term ``positional parameter''.

C.2.2 Quoting

Although this section contains a note indicating that a future version of this standard may extend the conditions under which some characters are special, there are no plans to do so. The note is there to encourage application writers to future-proof their shell code. In some cases existing widespread use of the characters unquoted would preclude them being given a special meaning in those use cases. For example, commas are in widespread use in filenames (notably by RCS and CVS) and it is common to pass the token `"{}"` as an argument to *find* and *xargs* unquoted.

Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

Austin Group Defects 1191 and 1193 are applied, adding:

```
]  ^  -  !  {  ,  }
```

to the list of characters that might need to be quoted under certain circumstances.

C.2.2.1 *Escape Character (Backslash)*

Austin Group Defect 500 is applied, changing ``follows'' to ``immediately follows''.

C.2.2.2 *Single-Quotes*

A <backslash> cannot be used to escape a single-quote in a single-quoted string. An embedded quote can be created by writing, for example: `"'a'\''b'"`, which yields `"a'b"`. (See XCU Section 2.6.5 (on page 2491) for a better understanding of how portions of words are either split into fields or remain concatenated.) A single token can be made up of concatenated partial strings containing all three kinds of quoting or escaping, thus permitting any combination of characters.

C.2.2.3 *Double-Quotes*

The escaped <newline> used for line continuation is removed entirely from the input and is not replaced by any white space. Therefore, it cannot serve as a token separator.

In double-quoting, if a <backslash> is immediately followed by a character that would be interpreted as having a special meaning, the <backslash> is deleted and the subsequent character is taken literally. If a <backslash> does not precede a character that would have a special meaning, it is left in place unmodified and the character immediately following it is also left unmodified. Thus, for example:

```
"\$"  ->  $
```

```
"\a"  ->  \a
```

It would be desirable to include the statement ``The characters from an enclosed `"${"` to the matching `'}'` shall not be affected by the double-quotes'', similar to the one for `"$()"`. However, historical practice in the System V shell prevents this.

Shell implementations differ widely in their handling of unescaped double-quote characters inside `"${...}"` (except for the four substring-processing variants). Hence this standard leaves the behavior unspecified. Single-quotes are ordinary characters in this context, and so cannot be used to quote a `'}'` within `"${...}"`. However, <backslash> can be used to escape a `'}'`. For example, the value of *foo* assigned by the following commands is `'}'`:

```
unset bar
foo="${bar-\}}"
```

When <backslash> is used in this way it is a special character and is therefore removed during quote removal, even though it would not be removed in:

```
foo="\}"
```

Differences in processing the `"${...}"` form led to inconsistencies between the historical System V shell, BSD, and KornShells, and the text in the Shell and Utilities volume of POSIX.1-2024 is an attempt to converge them without breaking too many applications. The only alternative to this compromise between shells would be to make the behavior unspecified not

just for unescaped double-quote but also for unescaped single-quote, '{', or '}'. The chosen requirements provide the maximum consistency between normal double-quote behavior and parameter expansion within double-quotes; the only real difference being the ability to escape a '}' with <backslash>.

Some implementations have allowed the end of the word to terminate the backquoted command substitution, such as in:

```
"`echo hello"
```

This usage is undefined; the matching backquote is required by the Shell and Utilities volume of POSIX.1-2024. The other undefined usage can be illustrated by the example:

```
sh -c '` echo "foo`'
```

The description of the recursive actions involving command substitution can be illustrated with an example. Upon recognizing the introduction of command substitution, the shell parses input (in a new context), gathering the source for the command substitution until an unbalanced ')' or '`' is located. For example, in the following:

```
echo "$(date; echo "
    one" )"
```

the double-quote following the *echo* does not terminate the first double-quote; it is part of the command substitution script. Similarly, in:

```
echo "$(echo *)"
```

the <asterisk> is not quoted since it is inside command substitution; however:

```
echo "$(echo "*")"
```

is quoted (and represents the <asterisk> character itself).

The $'...' construct does not retain its special meaning inside double quotes. This was discussed by the standard developers and rejected. Note that $'...' is a quoting mechanism and not an expansion. Losing the special meaning inside double-quotes is consistent with other quoting mechanisms losing their special meaning when quoted.

Austin Group Defect 221 is applied, clarifying the behavior of double-quotes within the string of characters from "${" to the matching '}' in parameter expansions using that form.

Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

Austin Group Defect 500 is applied, clarifying the behavior of <backslash> within double-quotes.

Austin Group Defect 1268 is applied, clarifying the effect of double-quotes on the results of parameter expansion, command substitution, or arithmetic expansion.

Austin Group Defect 1342 is applied, clarifying the requirements for alias substitutions inside command substitutions.

C.2.2.4 *Dollar-Single-Quotes*

The $'...' quoting construct has been implemented in several recent shells. It is similar to character string literals ("...") in the ISO C standard with the following exceptions:

- The \x escape sequence in C can be followed by an arbitrary number of hexadecimal digits. The *ksh93* implementation of $'...' also consumes an arbitrary number of hexadecimal digits; *bash* consumes at most two hexadecimal digits in this case. This standard leaves the result unspecified if more than two hexadecimal digits follow \x.

(Note that a hexadecimal escape followed by a literal hexadecimal character can always be represented as `$'\x`*XX*`'`*X*.)

- The `\c` escape sequence is not included in the ISO C standard. There was also some disagreement in shells that historically supported `\c` escape sequences in `$'...'`. These include:

  — Whether `\cA` through `\cZ` produced the byte values 1 through 26, respectively or supported the codeset independent control character as specified by the *stty* utility. This standard requires codeset independence.

  — Whether `\c[`, `\c\\`, `\c]`, `\c^`, `\c_`, and `\c?` could be used to yield the <ESC>, <FS>, <GS>, <RS>, <US>, and <DEL> control characters, respectively. This standard requires support for all of the control characters except NULL (matching what is done in the *stty* utility).

  — Whether `\c\\` or `\c\` was used to represent <FS>. This standard requires `\c\\` to make <backslash>-escape processing consistent.

  The implementors of the most common shells that implement `$'\c`*X*`'` agreed to convert to the behavior specified in this standard.

  Some shells also allow `\c`<arbitrary_control_character> to act as an inverse function to `\c`*X* (that is, `\cm` and `\cM` yield <CR> and `\c`<CR> yields `m` or `M`. This standard leaves this behavior implementation-defined.

- The `\e` escape sequence is not included in the ISO C standard, but was provided by all historical shells that supported `$'...'`. Some also supported `\E` as a synonym. One member of the group objected to adding `\e` because the <ESC> control character is not required to be in the portable character set. The `\e` sequence is included because many historical users of `$'...'` expect it to be there. The `\E` sequence is not included in this standard because <backslash>-escape sequences that start with <backslash> followed by an uppercase letter (except `\U`) are reserved by the ISO C standard for implementation use.

- The `\`*ddd* octal escape sequence and the `\x`*XX* hexadecimal escape sequence can be used to insert a null byte into a C character string literal and into a `$'...'` quoted word in this standard. In C, any characters specified after that null byte (including escape sequences) continue to be processed and added to the character string literal. In `$'...'` in the shell this standard allows the equivalent behavior but also allows the null byte and all remaining characters up to the terminating unescaped single-quote to be evaluated and discarded. The latter (which was historic practice in *bash*, but not in *ksh93*) allows an escape sequence producing a null byte to terminate the dollar-single-quoted expansion, but not terminate the token in which it appears if there are characters remaining in the token. For example:

  ```
  printf a$'b\0c\''d
  ```

  is required by this standard to produce:

  ```
  abd
  ```

  while historic versions of *ksh93* produced:

  ```
  ab
  ```

- The ISO C standard specifies `\u`*XXXX* and `\U`*XXXXXXXX* escape sequences. These need not be supported by `$'...'` in the shell. They were omitted because current shell implementations that support them differ in behavior. In particular, some shells always convert them to the UTF-8 encoding for the named character, even if the current locale's character set does not have UTF-8 encoding.

- The double-quote character can be used literally, while the single-quote character must be represented as an escape sequence. In C, single-quote can be used literally, while double-quote requires an escape sequence.
- A <backslash> immediately followed by a <newline> has unspecified behavior. In C, this sequence is used for line continuations, where both the <backslash> and <newline> are deleted and a diagnostic is required if a closing quote is not encountered before a <newline> that is not preceded by <backslash>. In current shell implementations, three different behaviors have been observed.
- The use of <backslash>-escape sequences not described in this standard results in unspecified behavior. In C, the result is not a token and a diagnostic is required. This allows shells to recognize other <backslash>-escape sequences in other ways as extensions to this standard. Furthermore, existing implementations already had different behaviors for some <backslash>-escape sequences when `$'...'` processing was added to this standard.

This standard makes the results implementation-defined if `\e` or `\c`*X* specifies a character that is not present in the current locale. Application authors should note that implementations are permitted to have a wide range of behaviors when encountering an unsupported character. For example:

- The shell might produce an error, possibly causing the shell to terminate.
- The unsupported character might be silently discarded.
- The unsupported character might be replaced with another character of a different character class.
- The unsupported character might be replaced with a shell-special character (e.g., `'?'`).
- The unsupported character might be replaced with multiple characters, shell-special or regular (e.g. if <ESC> is not supported, `$'\e'` may be replaced by `"???"`, `"XXX"`, or `"<ESC>"`).

However, implementations must document their behavior, and they are prohibited from replacing an unsupported character with bytes that do not form valid characters in the current locale's character set (e.g., encoding in UTF-8 when the locale has a 7-bit character set). This standard does not specify a way for script authors to determine beforehand whether a particular `\c`*X* sequence specifies a character that exists in the current locale. At the time this feature was standardized, no known implementations provided such a capability.

Note that the escape sequences recognized by `$'...'`, file format notation (see Table 5-1, on page 113), XSI-conforming implementations of the *echo* utility (see the utility's OPERANDS section in *echo*), and the *printf* utility's *format* operand (see the utility's EXTENDED DESCRIPTION in *printf*) are not the same. Some escape sequences are not recognized by all of the above, the `\c` escape sequence in *echo* is not at all like the `\c` escape sequence in `$'...'`, octal escape sequences in some of the above accept one to four octal digits and require a leading zero while others accept one to three octal digits and do not require a leading zero.

Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

C.2.3 Token Recognition

The "((" and "))" symbols are control operators in the KornShell, used for an alternative syntax of an arithmetic expression command. A conforming application cannot use "((" as a single token (with the exception of the "$((" form for shell arithmetic).

On some implementations, the symbol "((" is a control operator; its use produces unspecified results. Applications that wish to have nested subshells, such as:

```
((echo Hello);(echo World))
```

must separate the "((" characters into two tokens by including white space between them. Some systems may treat these as invalid arithmetic expressions instead of subshells.

Certain combinations of characters are invalid in portable scripts, as shown in the grammar. Implementations may use these combinations (such as "|&") as valid control operators. Portable scripts cannot rely on receiving errors in all cases where this volume of POSIX.1-2024 indicates that a syntax is invalid.

The (3) rule about combining characters to form operators is not meant to preclude systems from extending the shell language when characters are combined in otherwise invalid ways. Conforming applications cannot use invalid combinations, and test suites should not penalize systems that take advantage of this fact. For example, the unquoted combination "|&" is not valid in a POSIX script, but has a specific KornShell meaning.

The (10) rule about '#' as the current character is the first in the sequence in which a new token is being assembled. The '#' starts a comment only when it is at the beginning of a token. This rule is also written to indicate that the search for the end-of-comment does not consider escaped <newline> specially, so that a comment cannot be continued to the next line.

Because a *complete_command* encountered during a *program* is executed before the next *complete_command* is tokenized and parsed, syntax errors are not discovered by the shell until just before the code would be executed. While in some cases it might be desirable to detect and react to syntax errors before anything is executed (possible with *sh* –**n**), deferring the discovery of syntax errors has several benefits:

- It makes it possible for script authors to test for the availability of a nonstandard extension and react appropriately before the use of the extension would trigger a syntax error.
- It makes it possible to create self-extracting tarballs (a shell script concatenated with a payload archive that extracts the archive when executed).
- The shell does not have to read and parse the complete script before execution, which reduces memory usage when executing extremely long scripts.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0005 [718], XCU/TC2-2008/0006 [647], XCU/TC2-2008/0007 [568], and XCU/TC2-2008/0008 [648] are applied.

Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

Austin Group Defect 1036 is applied, clarifying how here-documents are parsed.

Austin Group Defect 1055 is applied, clarifying how much of a *program* is parsed before the parsed commands are executed.

Austin Group Defect 1083 is applied, changing ``the next character'' to ``each character in turn''.

Austin Group Defect 1085 is applied, clarifying requirements for the start and end of tokens.

C.2.3.1 Alias Substitution

The alias capability was added because it is widely used in historical implementations by interactive users.

The definition of ``alias name'' precludes an alias name containing a <slash> character. Since the text applies to the command words of simple commands, reserved words (in their proper places) cannot be confused with aliases.

The placement of alias substitution in token recognition makes it clear that it precedes all of the word expansion steps.

An example concerning trailing <blank> characters and reserved words follows. If the user types:

```
$ alias foo="/bin/ls "
$ alias while="/"
```

The effect of executing:

```
$ while true
> do
> echo "Hello, World"
> done
```

is a never-ending sequence of `"Hello, World"` strings to the screen. However, if the user types:

```
$ foo while
```

the result is an *ls* listing of **/**. Since the alias substitution for **foo** ends in a <space>, the next word is checked for alias substitution. The next word, **while**, has also been aliased, so it is substituted as well. Since it is not in the proper position as a command word, it is not recognized as a reserved word.

If the user types:

```
$ foo; while
```

while retains its normal reserved-word properties.

Some implementations add a <space> after the alias value when performing alias substitution in order to prevent the last character of the alias value and the first character after the alias name in the input from combining to form an operator. However, the extra <space> can have side-effects in other situations, such as if the alias value ends with an unquoted <backslash>. Implementations which do this are encouraged to change to an alternative method of delimiting a partial operator token at the end of an alias value.

Some, but not all, shell implementations do not process changes to alias definitions until the current *compound_list* (see XCU Section 2.10, on page 2512) has completed. In these shells, alias changes do not take effect until the end of the dot script, *eval* command, function invocation, **if** statement, **case** statement, **for** statement, **while** statement, or **until** statement containing the alias change.

Many shell implementations execute the contents of a file, typically **~/.profile**, when invoked as a login shell. The standard developers are unaware of any such implementations that process the contents of **~/.profile** (and similar startup files) as a single *compound_list*, so alias changes in **~/.profile** typically do take effect before the end of **~/.profile**.

Austin Group Defects 953 and 1630 are applied, providing additional detail on how alias substitution is performed.

C.2.4 Reserved Words

All reserved words are recognized syntactically as such in the contexts described. However, note that **in** is the only meaningful reserved word after a **case** or **for**; similarly, **in** is not meaningful as the first word of a simple command.

Reserved words are recognized only when they are delimited (that is, meet the definition of XBD Section 3.420, on page 93), whereas operators are themselves delimiters. For instance, `'('` and `')'` are control operators, so that no <space> is needed in (*list*). However, `'{'` and `'}'` are reserved words in { *list*;}, so that in this case the leading <space> and <semicolon> are required.

The list of unspecified reserved words is from the KornShell, so conforming applications cannot use them in places a reserved word would be recognized. Earlier versions of this standard omitted **time** from this list, so that the *time* utility could be included without requiring applications to quote all or part of its name (or use other measures) in order to avoid it being treated as a reserved word. However, although the intent was to allow the reserved word implementation (as evidenced by use of *time* in pipelines being unspecified, and explicit mention in the rationale of the *time* utility), the conditions under which the behavior was unspecified were insufficient to allow this. In particular, redirection in KornShell does not work in the normal way when **time** is a reserved word:

```
time utility 2> time.out
```

only writes the standard error from *utility* to **time.out**; the timing information is written to the shell's standard error, but these versions of the standard required the timing information to be written to **time.out**. Another issue was that if **time** is a reserved word, an application cannot define a function with that name, but these versions of the standard required that applications could do so. Hence **time** has now been added to the list of unspecified reserved words, but with its use as a reserved word limited in order to be compatible with its use as a utility in the cases where the two have traditionally had the same effect (other than possible output format differences).

There was a strong argument for promoting braces to operators (instead of reserved words), so they would be syntactically equivalent to subshell operators. Concerns about compatibility outweighed the advantages of this approach. Nevertheless, conforming applications should consider quoting `'{'` and `'}'` when they represent themselves.

When used in circumstances where reserved words are recognized, all words whose final character is a <colon> (`':'`) are reserved. The case of a name suffixed with a colon is reserved to allow implementations to support named labels for flow control; see the RATIONALE for the *break* special built-in utility. Other words ending in <colon> are reserved to provide implementations with a way to add new reserved words while still conforming to this standard.

It is possible that a future version of the Shell and Utilities volume of POSIX.1-2024 may require that `'{'` and `'}'` be treated individually as control operators, although the token `"{}"` will probably be a special-case exemption from this because of the often-used *find*{} construct.

Austin Group Defect 267 is applied, adding **time** to the list of words that may be recognized as reserved words while specifying its behavior if it is recognized as a reserved word, and extending the reservation of words whose final character is <colon> from those that are a name followed by a <colon> to all such words.

Austin Group Defect 465 is applied, adding **namespace** to the list of words that may be recognized as reserved words.

C.2.5 Parameters and Variables

Austin Group Defect 1561 is applied, clarifying that parameters can contain byte sequences that do not form valid characters and that the shell processes their values as characters only when performing operations that are described in this standard in terms of characters.

C.2.5.1 *Positional Parameters*

Austin Group Defect 1491 is applied, clarifying the handling of leading zeros in positional parameter identifiers.

C.2.5.2 *Special Parameters*

Most historical implementations implement subshells by forking; thus, the special parameter `'$'` does not necessarily represent the process ID of the shell process executing the commands since the subshell execution environment preserves the value of `'$'`.

If a subshell were to execute a background command, the value of `"$!"` for the parent would not change. For example:

```
(
date &
echo $!
)
echo $!
```

would echo two different values for `"$!"`.

The `"$-"` special parameter can be used to save and restore *set* options:

```
Save=$(echo $- | sed 's/[ics]//g')
...
set +aCefnuvx
if [ -n "$Save" ]; then
    set -$Save
fi
```

The three options are removed using *sed* in the example because they may appear in the value of `"$-"` (from the *sh* command line), but are not valid options to *set*.

The descriptions of parameters `'*'` and `'@'` assume the reader is familiar with the field splitting discussion in XCU Section 2.6.5 (on page 2491) and understands that portions of the word remain concatenated unless there is some reason to split them into separate fields.

The following examples illustrate some of the ways in which `'*'` and `'@'` can be expanded:

```
set "abc" "def ghi" "jkl"
unset novar
IFS=' ' # a space
printf '%s\n' $*
abc
def
ghi
jkl
printf '%s\n' "$*"
abc def ghi jkl
printf '%s\n' xx$*yy
xxabc
```



```
134316      def
134317      ghi
134318      jklyy
134319      printf '%s\n' "xx$*yy"
134320      xxabc def ghi jklyy
134321      printf '%s\n' $@
134322      abc
134323      def
134324      ghi
134325      jkl
134326      printf '%s\n' "$@"
134327      abc
134328      def ghi
134329      jkl
134330      printf '%s\n' ${1+"$@"}
134331      abc
134332      def ghi
134333      jkl
134334      printf '%s\n' ${novar-"$@"}
134335      abc
134336      def ghi
134337      jkl
134338      printf '%s\n' xx$@yy
134339      xxabc
134340      def
134341      ghi
134342      jklyy
134343      printf '%s\n' "xx$@yy"
134344      xxabc
134345      def ghi
134346      jklyy
134347      printf '%s\n' $@$@
134348      abc
134349      def
134350      ghi
134351      jklabc
134352      def
134353      ghi
134354      jkl
134355      printf '%s\n' "$@$@"
134356      abc
134357      def ghi
134358      jklabc
134359      def ghi
134360      jkl
134361      IFS=':'
134362      printf '%s\n' "$*"
134363      abc:def ghi:jkl
134364      var=$*; printf '%s\n' "$var"
134365      abc:def ghi:jkl
134366      var="$*"; printf '%s\n' "$var"
134367      abc:def ghi:jkl
134368      unset var
```



```
134369      printf '%s\n' ${var-$*}
134370      abc
134371      def ghi
134372      jkl
134373      printf '%s\n' "${var-$*}"
134374      abc:def ghi:jkl
134375      printf '%s\n' ${var-"$*"}
134376      abc:def ghi:jkl
134377      printf '%s\n' ${var=$*}
134378      abc
134379      def ghi
134380      jkl
134381      printf 'var=%s\n' "$var"
134382      var=abc:def ghi:jkl
134383      unset var
134384      printf '%s\n' "${var=$*}"
134385      abc:def ghi:jkl
134386      printf 'var=%s\n' "$var"
134387      var=abc:def ghi:jkl

134388      IFS='' # null
134389      printf '%s\n' "$*"
134390      abcdef ghijkl
134391      var=$*; printf '%s\n' "$var"
134392      abcdef ghijkl
134393      var="$*"; printf '%s\n' "$var"
134394      abcdef ghijkl
134395      unset var
134396      printf '%s\n' ${var-$*}
134397      abc
134398      def ghi
134399      jkl
134400      printf '%s\n' "${var-$*}"
134401      abcdef ghijkl
134402      printf '%s\n' ${var-"$*"}
134403      abcdef ghijkl
134404      printf '%s\n' ${var=$*}
134405      abcdef ghijkl
134406      printf 'var=%s\n' "$var"
134407      var=abcdef ghijkl
134408      unset var
134409      printf '%s\n' "${var=$*}"
134410      abcdef ghijkl
134411      printf 'var=%s\n' "$var"
134412      var=abcdef ghijkl
134413      printf '%s\n' "$@"
134414      abc
134415      def ghi
134416      jkl

134417      unset IFS
134418      printf '%s\n' "$*"
134419      abc def ghi jkl
134420      var=$*; printf '%s\n' "$var"
```



```
abc def ghi jkl
var="$*"; printf '%s\n' "$var"
abc def ghi jkl
unset var
printf '%s\n' ${var-$*}
abc
def
ghi
jkl
printf '%s\n' "${var-$*}"
abc def ghi jkl
printf '%s\n' ${var-"$*"}
abc def ghi jkl
printf '%s\n' ${var=$*}
abc
def
ghi
jkl
printf 'var=%s\n' "$var"
var=abc def ghi jkl
unset var
printf '%s\n' "${var=$*}"
abc def ghi jkl
printf 'var=%s\n' "$var"
var=abc def ghi jkl
printf '%s\n' "$@"
abc
def ghi
jkl

set one "" three
printf '[%s]\n' $*
[one]
[] (this line of output is optional)
[three]
printf '[%s]\n' $@
[one]
[] (this line of output is optional)
[three]

set --
printf '[%s]\n' foo "$*"
[foo]
[]
printf '[%s]\n' foo "$novar$*$(echo)"
[foo]
[]
printf '[%s]\n' foo $@
[foo]
printf '[%s]\n' foo "$@"
[foo]
printf '[%s]\n' foo ''$@
[foo]
[]
```



```
printf '[%s]\n' foo '"$@"
[foo]
[]
printf '[%s]\n' foo "$novar$@$(echo)"
[foo]
[] (this line of output is optional)
printf '[%s]\n' foo '"$novar$@$(echo)"
[foo]
[]
```

In all of the following commands the results of the expansion of '@' (if performed) are unspecified:

```
var=$@
var="$@"
printf '%s\n' ${var=$@}
printf '%s\n' "${var=$@}"
printf '%s\n' ${var="$@"}
printf '%s\n' ${var?$@}
printf '%s\n' "${var?$@}"
printf '%s\n' ${var?"$@"}
printf '%s\n' ${#@}
printf '%s\n' "${#@}"
printf '%s\n' ${@%foo}
printf '%s\n' "${@%foo}"
printf '%s\n' ${@#foo}
printf '%s\n' "${@#foo}"
printf '%s\n' ${var%$@}
printf '%s\n' "${var%$@}"
printf '%s\n' ${var%"$@"}
printf '%s\n' ${var%%$@}
printf '%s\n' "${var%%$@}"
printf '%s\n' ${var%%"$@"}
printf '%s\n' ${var#$@}
printf '%s\n' "${var#$@}"
printf '%s\n' ${var#"$@"}
printf '%s\n' ${var##$@}
printf '%s\n' "${var##$@}"
printf '%s\n' ${var##"$@"}
```

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0009 [888] is applied.

Austin Group Defect 1039 is applied, clarifying the description of the '-' special parameter.

Austin Group Defect 1052 is applied, clarifying that decimal valued special parameters expand to the shortest representation.

Austin Group Defects 1150 and 1309 are applied, clarifying the description of the '?' special parameter.

Austin Group Defect 1254 is applied, clarifying how the '?' and '!' special parameters are affected by job control.

C.2.5.3 *Shell Variables*

Since shell variables are parameters denoted by a name, the shell cannot initialize shell variables from environment variables that do not have a valid name. However, the shell may initialize parameters that do not have valid names from such environment variables.

See the discussion of *IFS* in Section C.2.6.5 (on page 3889) and the RATIONALE for the *sh* utility.

The prohibition on *LC_CTYPE* changes affecting lexical processing protects the shell implementor (and the shell programmer) from the ill effects of changing the definition of <blank> or the set of alphabetic characters in the current environment. It would probably not be feasible to write a compiled version of a shell script without this rule. The rule applies only to the current invocation of the shell and its subshells—invoking a shell script or performing *exec sh* would subject the new shell to the changes in *LC_CTYPE*.

Other common environment variables used by historical shells are not specified by the Shell and Utilities volume of POSIX.1-2024, but they should be reserved for the historical uses.

Tilde expansion for components of *PATH* in an assignment such as:

```
PATH=~hlj/bin:~dwc/bin:$PATH
```

is a feature of some historical shells and is allowed by the wording of XCU Section 2.6.1 (on page 2485). Note that the <tilde> characters are expanded during the assignment to *PATH*, not when *PATH* is accessed during command search.

The following entries represent additional information about variables included in the Shell and Utilities volume of POSIX.1-2024, or rationale for common variables in use by shells that have been excluded:

_ (Underscore.) While <underscore> is historical practice, its overloaded usage in the KornShell is confusing, and it has been omitted from the Shell and Utilities volume of POSIX.1-2024.

ENV This variable can be used to set aliases and other items local to the invocation of a shell. The file referred to by *ENV* differs from **$HOME/.profile** in that **.profile** is typically executed at session start-up, whereas the *ENV* file is executed at the beginning of each shell invocation. The *ENV* value is interpreted in a manner similar to a dot script, in that the commands are executed in the current environment and the file needs to be readable, but not executable. However, unlike dot scripts, no *PATH* searching is performed. This is used as a guard against Trojan Horse security breaches.

ERRNO This variable was omitted from the Shell and Utilities volume of POSIX.1-2024 because the values of error numbers are not defined in POSIX.1-2024 in a portable manner.

FCEDIT Since this variable affects only the *fc* utility, it has been omitted from this more global place. The value of *FCEDIT* does not affect the command-line editing mode in the shell; see the description of *set* **–o** *vi* in the *set* built-in utility.

PS1 This variable is used for interactive prompts. Historically, the ``superuser'' has had a prompt of `'#'`. Since privileges are not required to be monolithic, it is difficult to define which privileges should cause the alternate prompt. However, a sufficiently powerful user should be reminded of that power by having an alternate prompt.

PS3 This variable is used by the KornShell for the *select* command. Since the POSIX shell does not include *select*, *PS3* was omitted.

PS4 This variable is used for shell debugging. For example, the following script:

```
PS4='[${LINENO}]+ '
set -x
echo Hello
```

writes the following to standard error:

```
[3]+ echo Hello
```

RANDOM This pseudo-random number generator was not seen as being useful to interactive users.

SECONDS Although this variable is sometimes used with *PS1* to allow the display of the current time in the prompt of the user, it is not one that would be manipulated frequently enough by an interactive user to include in the Shell and Utilities volume of POSIX.1-2024.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0002 [152] is applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0010 [888], XCU/TC2-2008/0011 [884], and XCU/TC2-2008/0012 [494] are applied.

Austin Group Defect 953 is applied, clarifying how the *ENV* file is parsed.

Austin Group Defect 1006 is applied, clarifying how the values of the *PS1*, *PS2*, and *PS4* variables are expanded.

Austin Group Defect 1441 is applied, requiring *PS4* to be used in non-interactive shells.

Austin Group Defect 1511 is applied, making the description of *LINENO* consistent with other variables as regards how they relate to the User Portability Utilities option.

Austin Group Defect 1561 is applied, clarifying that shell variables are initialized only from environment variables that have valid names.

C.2.6 Word Expansions

Some shells implement *brace expansion* which expands, for example, `file{A,B,C}.c` into the fields `fileA.c`, `fileB.c`, and `fileC.c` or `file{1..3}.c` into the fields `file1.c`, `file2.c`, and `file3.c`. This form of expansion is allowed but not required by this standard, but if supported must be performed before all of the standard word expansions. A variant which some shells implement whereby brace expansion is performed following field splitting was considered by the standard developers and rejected because it causes surprising behavior if the results of parameter expansion and command substitution happen to produce a valid brace expansion. For example, if the shell variable `patt` contains an arbitrary pathname, glob pattern applications cannot rely on `some_command -- $patt` passing a list of pathnames that match the pattern to `some_command`. Note that quoting the braces or commas prevents this form of expansion, but quoting the periods need not prevent it.

Step (2) refers to the ``portions of fields generated by step (1)''. For example, if the word being expanded were `"$x+$y"` and *IFS*=+, the word would be split only if `"$x"` or `"$y"` contained `'+'`; the `'+'` in the original word was not generated by step (1).

IFS is used for performing field splitting on the results of parameter and command substitution; it is not used for splitting all fields. Earlier versions of the shell used it for splitting all fields during field splitting, but this has severe problems because the shell can no longer parse its own script. There are also important security implications caused by this behavior. All useful applications of *IFS* use it for parsing input of the *read* utility and for splitting the results of

134606 parameter and command substitution.

134607 The rule concerning expansion to a single field requires that if **foo**=**abc** and **bar**=**def**, that:

134608
```
"$foo""$bar"
```

134609 expands to the single field:

134610
```
abcdef
```

134611 The rule concerning empty fields can be illustrated by:

134612
```
$    unset foo
$    set $foo bar '' xyz "$foo" abc
$    for i
>    do
>        echo "-$i-"
>    done
-bar-
--
-xyz-
--
-abc-
```

134623 Step (1) indicates that parameter expansion, command substitution, and arithmetic expansion
134624 are all processed simultaneously as they are scanned. For example, the following is valid
134625 arithmetic:

134626
```
x=1
echo $(( $(echo 3)+$x ))
```

134628 An early proposal stated that tilde expansion preceded the other steps, but this is not the case in
134629 known historical implementations; if it were, and if a referenced home directory contained a `'$'`
134630 character, expansions would result within the directory name.

134631 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0003 [49,430] is applied.

134632 Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

134633 Austin Group Defect 985 is applied, clarifying that quote removal is not always performed.

134634 Austin Group Defect 1038 is applied, clarifying that a `'$'` that is followed by a <space>, <tab>,
134635 or a <newline>, or is not followed by any character, is treated as a literal character.

134636 Austin Group Defect 1123 is applied, clarifying the environment in which expansions are
134637 performed and requirements relating to empty fields.

134638 Austin Group Defect 1193 is applied, adding optional brace expansion.

134639 *C.2.6.1 Tilde Expansion*

134640 Tilde expansion generally occurs only at the beginning of words, but an exception based on
134641 historical practice has been included:

134642
```
PATH=/posix/bin:~dgk/bin
```

134643 This is eligible for tilde expansion because <tilde> follows a <colon> and none of the relevant
134644 characters is quoted. Consideration was given to prohibiting this behavior because any of the
134645 following are reasonable substitutes:

134646
```
PATH=$(printf %s ~karels/bin : ~bostic/bin)
```



```
for Dir in ~maart/bin ~srb/bin ...
do
    PATH=${PATH:+$PATH:}$Dir
done
```

In the first command, explicit <colon> characters are used for each directory. In all cases, the shell performs tilde expansion on each directory because all are separate words to the shell.

Note that expressions in operands such as:

```
make -k mumble LIBDIR=~chet/lib
```

do not qualify as shell variable assignments, and tilde expansion is not performed (unless the command does so itself, which *make* does not).

Because of the requirement that the word is not quoted, the following are not equivalent; only the last causes tilde expansion:

```
\~hlj/   ~h\lj/   ~"hlj"/   ~hlj\/   ~hlj/
```

In an early proposal, tilde expansion occurred following any unquoted <equals-sign> or <colon>, but this was removed because of its complexity and to avoid breaking commands such as:

```
rcp hostname:~marc/.profile .
```

System administrators on systems where **//** has an implementation-defined meaning which is different to **/**, should not create users with a home directory of **/** or **//**, since this may lead to unexpected filename resolution on those systems.

A suggestion was made that the special sequence `"$~"` should be allowed to force tilde expansion anywhere. Since this is not historical practice, it has been left for future implementations to evaluate. (The description in XCU Section 2.2 (on page 2472) requires that a <dollar-sign> be quoted to represent itself, so the `"$~"` combination is already unspecified.)

The results of giving <tilde> with an unknown login name are undefined because the KornShell `"~+"` and `"~-"` constructs make use of this condition, but in general it is an error to give an incorrect login name with <tilde>. The results of having *HOME* unset are unspecified because some historical shells treat this as an error.

Historically, the Korn shell performed field splitting and pathname expansion on the results of tilde expansion, and earlier versions of this standard reflected this. However, tilde expansion results in a pathname, and performing field splitting and pathname expansion on something that is already a pathname is at best redundant and at worst will change the value from the correct pathname to one or more incorrect ones. Later versions of the Korn shell do not perform these expansions and POSIX.1-2024 has been updated to match. Note that although pathname expansion is not performed on the results of tilde expansion, this does not prevent other parts of the same word from being expanded. For example, `~/a*` expands to all files in **$HOME** beginning with `'a'`.

Austin Group Defect 1172 is applied, clarifying how quoting affects tilde expansion.

Austin Group Defect 1632 is applied, clarifying the treatment of <slash> characters in tilde expansion.

C.2.6.2 *Parameter Expansion*

The rule for finding the closing `'}'` in `"${...}"` is the one used in the KornShell and is upwardly-compatible with the Bourne shell, which does not determine the closing `'}'` until the word is expanded. The advantage of this is that incomplete expansions, such as:

```
${foo
```

can be determined during tokenization, rather than during expansion.

Quote removal is performed when assigning the value in the ${*parameter*:=[*word*]} form of expansion in order that a subsequent expansion of the same parameter produces the same value as the original expansion. That is, the commands:

```
unset parameter
foo=${parameter:=word}
bar=${parameter}
```

assign the same value to *foo* and *bar*. A consequence of this is that the expansions ${*parameter*:=[*word*]} and ${*parameter*:-[*word*]} can produce different results for the same *word*. For example, with *parameter* unset or empty:

```
${parameter:-a\ b}
```

expands to a single field "a b", whereas:

```
${parameter:=a\ b}
```

expands to two fields `'a'` and `'b'` (because *parameter* is assigned the value `"a b"` before its value is substituted).

For rationale regarding expansion of `"${...}"` within double-quotes, see Section C.2.2.3 (on page 3867).

The string length and substring capabilities were included because of the demonstrated need for them, based on their usage in other shells, such as C shell and KornShell.

Historical versions of the KornShell have not performed tilde expansion on the word part of parameter expansion; however, it is more consistent to do so.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0004 [458], XCU/TC1-2008/0005 [458], XCU/TC1-2008/0006 [457], XCU/TC1-2008/0007 [457], XCU/TC1-2008/0008 [417], XCU/TC1-2008/0009 [457], XCU/TC1-2008/0010 [457], XCU/TC1-2008/0011 [457], XCU/TC1-2008/0012 [457], XCU/TC1-2008/0013 [457], XCU/TC1-2008/0014 [457], XCU/TC1-2008/0015 [457], XCU/TC1-2008/0016 [457], XCU/TC1-2008/0017 [457], and XCU/TC1-2008/0018 [458] are applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0013 [888] and XCU/TC2-2008/0014 [867] are applied.

Austin Group Defect 221 is applied, removing a statement about counting brace levels and clarifying that quote removal is performed when expanding *word* in ${*parameter*:=[*word*]}.

Austin Group Defect 985 is applied, clarifying when quote removal is performed.

Austin Group Defect 1052 is applied, clarifying the description of string length expansion.

Austin Group Defect 1268 is applied, removing text relating to parameter expansion inside double-quotes.

Austin Group Defect 1478 is applied, making explicitly unspecified the results of parameter expansions that test whether the parameter `'*'` or `'@'` is unset or null.

Austin Group Defect 1491 is applied, restructuring a paragraph that used ``Otherwise'' after two

134730 conditions.

134731 Austin Group Defect 1561 is applied, clarifying that the varieties of parameter expansion that
134732 provide for substring processing process parameter values as characters.

134733 *C.2.6.3 Command Substitution*

134734 The "$()" form of command substitution solves a problem of inconsistent behavior when using
134735 backquotes. For example:

134736

Command	Output
`echo '\$x'`	`\$x`
`echo `echo '\$x'``	`$x`
`echo $(echo '\$x')`	`\$x`

134737
134738
134739

134740 Additionally, the backquoted syntax has historical restrictions on the contents of the embedded
134741 command. While the newer "$()" form can process any kind of valid embedded script (with a
134742 few caveats; see below), the backquoted form cannot handle some valid scripts that include
134743 backquotes. For example, these otherwise valid embedded scripts do not work in the left
134744 column, but do work on the right:

```
134745 echo `                              echo $(
134746 cat <<\eof                          cat <<\eof
134747 a here-doc with `                   a here-doc with )
134748 eof                                 eof
134749 `                                   )

134750 echo `                              echo $(
134751 echo abc # a comment with `         echo abc # a comment with )
134752 `                                   )

134753 echo `                              echo $(
134754 echo '`'                           echo ')'
134755 `                                   )
```

134756 Because of these inconsistent behaviors, the backquoted variety of command substitution is not
134757 recommended for new applications that nest command substitutions or attempt to embed
134758 complex scripts.

134759 The KornShell feature:

134760 > If the *commands* string is of the form **<***word*, *word* is expanded to generate a pathname, and
134761 > the value of the command substitution is the contents of this file with any trailing
134762 > <newline> characters deleted.

134763 was omitted from the Shell and Utilities volume of POSIX.1-2024 because $(*cat word*) is an
134764 appropriate substitute. However, to prevent breaking numerous scripts relying on this feature, it
134765 is unspecified to have a script within "$()" that has only redirections.

134766 In IEEE Std 1003.2-1992 the $(*commands*) form of command substitution only had unspecified
134767 behavior for a *commands* string consisting solely of redirections. However, two additional
134768 unspecified cases have since been added with relation to aliases:

134769 1. Implementations are permitted to parse the entire *commands* string before executing any
134770 of it, and in this case *alias* and *unalias* commands in *commands* have no effect during
134771 parsing. For example, the following commands:

134772 ```
alias foo='echo "hello globe"'
```

  
```

```
echo $(alias foo='echo "Hello World"';foo)
```

produce the output `"hello globe"` if the *commands* string is executed as an entire command and produce the output `"Hello World"` if the *commands* string is executed incrementally.

2. Although existing aliases are required to be expanded when the shell parses the input that follows the `"$("` in order to find the terminating `')'` (see Section 2.3, on page 2475), it is unspecified whether the terminating `')'` can result from alias substitution. For example, with this script:

```
alias foo="echo foo )"
echo $(foo ; echo bar
```

some shells output lines containing `"foo"` and `"bar"` whereas other shells report a syntax error because they do not find a terminating `')'` for the command substitution.

Arithmetic expansions have precedence over command substitutions. That is, if the shell can parse an expansion beginning with `"$(("` as an arithmetic expansion then it will do so. It will only parse the expansion as a command substitution (that starts with a subshell) if it determines that it cannot parse the expansion as an arithmetic expansion. If the syntax is valid for neither type of expansion, then it is unspecified what kind of syntax error the shell reports.

How well the shell performs this determination is a quality of implementation issue. Current shell implementations use heuristics. In particular, the shell need not evaluate nested expansions when determining whether it can parse an expansion beginning with `"$(("` as an arithmetic expansion. For example:

```
$((a $op b))
```

is always an arithmetic expansion if `"$op"` expands to, say, `'+'`, but if `"$op"` expands to `'('` then the shell might still parse the expansion as an arithmetic expansion (resulting in a syntax error due to unbalanced parentheses) or it might perform a command substitution.

This standard requires that conforming applications always separate the `"$("` and `'('` with white space when a command substitution starts with a subshell. This is because implementations may support extensions in arithmetic expressions which could result in the shell parsing the input as an arithmetic expansion even though a minimally conforming shell would not. For example, many shells support arrays with the array index (which can be an expression) in square brackets. Therefore, the presence of `"myfile[0-9]"` within an expansion beginning `"$(("` is no guarantee that it will be parsed as a command substitution.

The ambiguity is not restricted to the simple case of a single subshell. More complicated ambiguous cases are possible (even with just the standard shell syntax), such as:

```
$(( cat <<EOH
+ ( (
EOH
) && ( cat <<EOH
) ) + 1 +
EOH
))
```

This can be parsed as an arithmetic expansion, with *cat* and *EOH* as the names of shell variables. Ambiguous cases also exist where the end of the expansion is at a different location for the arithmetic expansion and the command substitution:

```
$((cat <<EOF
+((((
```



```
EOF
) && (
cat <<EOF
+
EOF
))
```

This is an incomplete arithmetic expansion, but would have been a (complete) command substitution if it could not have been parsed as an arithmetic expansion. If this expansion occurs at the end of input then the shell reports a syntax error; it does not parse it as a command substitution.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/4 is applied, changing the text from: ``If a command substitution occurs inside double-quotes, it shall not be performed on the results of the substitution.'' to: ``If a command substitution occurs inside double-quotes, field splitting and pathname expansion shall not be performed on the results of the substitution.''. The replacement text taken from the ISO POSIX-2:1993 standard is clearer about the items that are not performed.

SD5-XCU-ERN-84 is applied, clarifying how the search for the matching backquote is satisfied.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0019 [217] is applied.

Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

Austin Group Defect 953 is applied, clarifying how the commands in command substitutions are parsed.

Austin Group Defect 1015 is applied, clarifying the handling of <backslash> when a backquoted command substitution is within double-quotes.

Austin Group Defect 1268 is applied, removing text relating to command substitution inside double-quotes.

Austin Group Defect 1342 is applied, clarifying the requirements for alias substitutions inside command substitutions.

Austin Group Defect 1560 is applied, clarifying that the standard output of the command(s) in a command substitution is treated as a sequence of bytes.

C.2.6.4 *Arithmetic Expansion*

The standard developers agreed that there was a strong desire for some kind of arithmetic evaluator to provide functionality similar to *expr*, that relating it to `'$'` makes it work well with the standard shell language and provides access to arithmetic evaluation in places where accessing a utility would be inconvenient.

The syntax and semantics for arithmetic were revised for the ISO/IEC 9945-2:1993 standard. The language represents a simple subset of the previous arithmetic language (which was derived from the KornShell `"(())"` construct). The syntax was changed from that of a command denoted by ((*expression*)) to an expansion denoted by $((*expression*)). The new form is a dollar expansion (`'$'`) that evaluates the expression and substitutes the resulting value. Objections to the previous style of arithmetic included that it was too complicated, did not fit in well with the use of variables in the shell, and its syntax conflicted with subshells. The justification for the new syntax is that the shell is traditionally a macro language, and if a new feature is to be added, it should be accomplished by extending the capabilities presented by the current model of the shell, rather than by inventing a new one outside the model; adding a new dollar expansion was perceived to be the most intuitive and least destructive way to add such a

new capability.

The standard requires assignment operators to be supported (as listed in XCU Section 1.1.2, on page 2457), and since arithmetic expansions are not specified to be evaluated in a subshell environment, changes to variables there have to be in effect after the arithmetic expansion, just as in the parameter expansion `"${x=value}"`.

Note, however, that `"$(( x=5 ))"` need not be equivalent to `"$(( $x=5 ))"`. If the value of the environment variable *x* is the string `"y="`, the expansion of `"$(( x=5 ))"` would set *x* to 5 and output 5, but `"$(( $x=5 ))"` would output 0 if the value of the environment variable *y* is not 5 and would output 1 if the environment variable *y* is 5. Similarly, if the value of the environment variable is 4, the expansion of `"$(( x=5 ))"` would still set *x* to 5 and output 5, but `"$(( $x=5 ))"` (which would be equivalent to `"$(( 4=5 ))"`) would yield a syntax error.

In early proposals, a form $[*expression*] was used. It was functionally equivalent to the `"$(())"` of the current text, but objections were lodged that the 1988 KornShell had already implemented `"$(())"` and there was no compelling reason to invent yet another syntax. Furthermore, the `"$[]"` syntax had a minor incompatibility involving the patterns in **case** statements.

The portion of the ISO C standard arithmetic operations selected corresponds to the operations historically supported in the KornShell. In addition to the exceptions listed in XCU Section 2.6.4 (on page 2490), the use of the following are explicitly outside the scope of the rules defined in XCU Section 1.1.2.1 (on page 2457):

- The prefix operator `'&'` and the `"[]"`, `"->"`, and `'.'` operators.
- Casts

It was concluded that the *test* command ([) was sufficient for the majority of relational arithmetic tests, and that tests involving complicated relational expressions within the shell are rare, yet could still be accommodated by testing the value of `"$(())"` itself. For example:

```
# a complicated relational expression
while [ $(( (($x + $y)/($a * $b)) < ($foo*$bar) )) -ne 0 ]
```

or better yet, the rare script that has many complex relational expressions could define a function like this:

```
val() {
    return $((!$1))
}
```

and complicated tests would be less intimidating:

```
while val $(( (($x + $y)/($a * $b)) < ($foo*$bar) ))
do
    # some calculations
done
```

A suggestion that was not adopted was to modify *true* and *false* to take an optional argument, and *true* would exit true only if the argument was non-zero, and *false* would exit false only if the argument was non-zero:

```
while true $(($x > 5 && $y <= 25))
```

There is a minor portability concern with the new syntax. The example `"$((2+2))"` could have been intended to mean a command substitution of a utility named `"2+2"` in a subshell. The standard developers considered this to be obscure and isolated to some KornShell scripts (because `"$()"` command substitution existed previously only in the KornShell). The text on

134909 command substitution requires that the `"$("` and `'('` be separate tokens if this usage is
134910 needed.

134911 An example such as:

134912 ```
echo $((echo hi);(echo there))
```

134913 should not be misinterpreted by the shell as arithmetic because attempts to balance the
134914 parentheses pairs would indicate that they are subshells. However, as indicated by XBD Section
134915 3.85 (on page 44), a conforming application must separate two adjacent parentheses with white
134916 space to indicate nested subshells.

134917 The standard is intentionally silent about how a variable's numeric value in an expression is
134918 determined from its normal ``sequence of bytes'' value. It could be done as a text substitution, as
134919 a conversion like that performed by *strtol*(), or even recursive evaluation. Therefore, the only
134920 cases for which the standard is clear are those for which both conversions produce the same
134921 result. The cases where they give the same result are those where the sequence of bytes form a
134922 valid integer constant. Therefore, if a variable does not contain a valid integer constant, the
134923 behavior is unspecified.

134924 For the commands:

134925 ```
x=010; echo $((x += 1))
```

134926 the output must be 9.

134927 For the commands:

134928 ```
x=' 1'; echo $((x += 1))
```

134929 the results are unspecified.

134930 For the commands:

134931 ```
x=1+1; echo $((x += 1))
```

134932 the results are unspecified.

134933 Although the ISO C standard requires support for **long long** and allows extended integer types
134934 with higher ranks, POSIX.1-2024 only requires arithmetic expansions to support **signed long**
134935 integer arithmetic. Implementations are encouraged to support signed integer values at least as
134936 large as the size of the largest file allowed on the implementation.

134937 Implementations are also allowed to perform floating-point evaluations as long as an
134938 application won't see different results for expressions that would not overflow **signed long**
134939 integer expression evaluation. (This includes appropriate truncation of results to integer values.)

134940 Changes made in response to IEEE PASC Interpretation 1003.2 #208 removed the requirement
134941 that the integer constant suffixes `l` and `L` had to be recognized. The ISO POSIX-2: 1993 standard
134942 did not require the `u`, `ul`, `uL`, `U`, `Ul`, `UL`, `lu`, `lU`, `Lu`, and `LU` suffixes since only signed integer
134943 arithmetic was required. Since all arithmetic expressions were treated as handling **signed long**
134944 integer types anyway, the `l` and `L` suffixes were redundant. No known scripts used them and
134945 some historic shells did not support them. When the ISO/IEC 9899: 1999 standard was used as
134946 the basis for the description of arithmetic processing, the `ll` and `LL` suffixes and combinations
134947 were also not required. Implementations are still free to accept any or all of these suffixes, but
134948 are not required to do so.

134949 There was also some confusion as to whether the shell was required to recognize character
134950 constants. Syntactically, character constants were required to be recognized, but the
134951 requirements for the handling of <backslash> and single-quote characters (needed to specify
134952 character constants) within an arithmetic expansion were ambiguous. Furthermore, no known

shells supported them. Changes made in response to IEEE PASC Interpretation 1003.2 #208 removed the requirement to support them (if they were indeed required before). POSIX.1-2024 clearly does not require support for character constants.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/3 is applied, clarifying arithmetic expressions.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0020 [50] is applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0015 [584] is applied.

C.2.6.5 *Field Splitting*

The operation of field splitting using *IFS*, as described in early proposals, was based on the way the KornShell splits words, but it is incompatible with other common versions of the shell. However, each has merit, and so a decision was made to allow both. If the *IFS* variable is unset or is <space><tab><newline>, the operation is equivalent to the way the System V shell splits words. Using characters outside the <space><tab><newline> set yields the KornShell behavior, where each of the non-<space><tab><newline>s is significant. This behavior, which affords the most flexibility, was taken from the way the original *awk* handled field splitting.

The different handling of white space and non-white-space characters in *IFS* can be summarized as a pseudo-ERE:

```
(s*ns*|s+)
```

where *s* is an *IFS* white-space character and *n* is a character in the *IFS* that is not white space. Any string matching that ERE delimits a field, except that the *s*+ form does not delimit fields at the beginning or the end of a line. For example, if *IFS* is <space>/<comma>/<tab>, the string:

```
<space><space>red<space><space>,<space>white<space>blue
```

yields the three colors as the delimited fields.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0016 [832] is applied.

Austin Group Defect 1123 is applied, clarifying the requirements if no fields are delimited.

Austin Group Defect 1560 is applied, clarifying that the results of word expansions are treated as a sequences of bytes when searching for (bytes that form) *IFS* characters.

Austin Group Defect 1649 is applied, clarifying how field splitting is performed.

C.2.6.6 *Pathname Expansion*

There is no additional rationale provided for this section.

C.2.6.7 *Quote Removal*

The golden rule in quote removal is that if a quote character was treated as special in the original word, it is removed; if it was treated as a literal character, it is not removed.

Austin Group Defect 221 is applied, clarifying the conditions under which quote characters are, or are not, removed.

Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

C.2.7 Redirection

In the System Interfaces volume of POSIX.1-2024, file descriptors are integers in the range 0–({OPEN_MAX}–1). The file descriptors discussed in XCU Section 2.7 (on page 2493) are that same set of small integers.

Having multi-digit file descriptor numbers for I/O redirection can cause some obscure compatibility problems. Specifically, scripts that depend on an example command:

```
echo 22>/dev/null
```

echoing `"2"` to standard error or `"22"` to standard output are no longer portable. However, the file descriptor number must still be delimited from the preceding text. For example:

```
cat file2>foo
```

writes the contents of **file2**, not the contents of **file**.

The limitation to 9 file descriptors is overcome in some shells via a form of redirection whereby a shell variable stores the file descriptor number. For example:

```
exec {fdvar}> foo
```

opens the file `foo` on a file descriptor greater than 9 and stores the file descriptor number in shell variable `fdvar`. (This can later be closed using `exec {fdvar}>&-`.) This form of redirection is allowed but not required by this standard.

The `">|"` format of output redirection was adopted from the KornShell. Along with the *noclobber* option, *set* **–C**, it provides a safety feature to prevent inadvertent overwriting of existing files. (See the RATIONALE for the *pathchk* utility for why this step was taken.) The restriction on regular files is historical practice.

The System V shell and the KornShell have differed historically on pathname expansion of *word*; the former never performed it, the latter only when the result was a single field (file). As a compromise, it was decided that the KornShell functionality was useful, but only as a shorthand device for interactive users. No reasonable shell script would be written with a command such as:

```
cat foo > a*
```

Thus, shell scripts are prohibited from doing it, while interactive users can select the shell with which they are most comfortable.

The construct `"2>&1"` is often used to redirect standard error to the same file as standard output. Since the redirections take place beginning to end, the order of redirections is significant. For example:

```
ls > foo 2>&1
```

directs both standard output and standard error to file **foo**. However:

```
ls 2>&1 > foo
```

only directs standard output to file **foo** because standard error was duplicated as standard output before standard output was directed to file **foo**.

Applications should not use the `[`*n*`]<&-` or `[`*n*`]>&-` operators to execute a utility or application with file descriptor 0 not open for reading or with file descriptor 1 or 2 not open for writing, as this might cause the executed program (or shell built-in) to misbehave. In order not to pass on these file descriptors to an executed utility or application, applications should not just close them but should reopen them on, for example, **/dev/null**. Some implementations may reopen them automatically, but applications should not rely on this being done.

The "<>" operator could be useful in writing an application that worked with several terminals, and occasionally wanted to start up a shell. That shell would in turn be unable to run applications that run from an ordinary controlling terminal unless it could make use of "<>" redirection. The specific example is a historical version of the pager *more*, which reads from standard error to get its commands, so standard input and standard output are both available for their usual usage. There is no way of saying the following in the shell without "<>":

```
cat food | more - >/dev/tty03 2<>/dev/tty03
```

Another example of "<>" is one that opens **/dev/tty** on file descriptor 3 for reading and writing:

```
exec 3<> /dev/tty
```

An example of creating a lock file for a critical code region:

```
set -C
until    2> /dev/null > lockfile
do       sleep 30
done
set +C
perform critical function
rm lockfile
```

Since **/dev/null** is not a regular file, no error is generated by redirecting to it in *noclobber* mode.

Tilde expansion is not performed on a here-document because the data is treated as if it were enclosed in double-quotes.

Austin Group Defect 1193 is applied, adding the optional redirection form **{***location***}***redir-op word*.

Austin Group Defect 1232 is applied, clarifying the allowed behaviors in an interactive shell when pathname expansion on the word following a redirection operator would result in more than one word.

Austin Group Defect 1493 is applied, moving some information from this section to the definition of ``file descriptor'' in XBD Section 3.141 (on page 51).

C.2.7.1 *Redirecting Input*

There is no additional rationale provided for this section.

C.2.7.2 *Redirecting Output*

Earlier versions of this standard did not require redirection using '>' when *noclobber* is set to perform the file creation step as an atomic operation. Historical shells just called *stat*() to check if a regular file existed and then called *creat*(). The operation thus involved a race condition which meant that it could not be used for reliable creation of lock files. Many shell implementations improved on this by using *open*() with the O_CREAT and O_EXCL flags set as one step in a multi-step process which still meant that an existing non-regular file (for example **/dev/null**, **/dev/tty**, or a FIFO) was opened successfully. However, the methods employed still involved a race condition and could produce misleading diagnostics if there is concurrent creation or removal of files.

An ideal solution would be an O_NOCLOBBER flag for *open*() which the shell could use in order to perform the entire operation atomically, and implementations are encouraged to adopt this solution, adding the flag as described in the FUTURE DIRECTIONS section of *open*() (on page 1515), and using it in the implementation's POSIX shell and in other shells. Authors of

135075 portable shells should make use of `#ifdef O_NOCLOBBER` so that it is used on
135076 implementations that provide it.

135077 If O_NOCLOBBER is not used, shells can use one of the following methods:

135078 1. The ``stat first'' method.

135079 a. Call *stat*( ) and if the file exists and is a regular file, the redirection fails. Otherwise:

135080 b. Call *open*( ) without O_CREAT or O_TRUNC to open an existing file. If the open
135081 succeeds, use *fstat*( ) to check whether the opened file is a regular file. If it is, close
135082 it and fail the redirection. If it is a non-regular file, the redirection succeeds.
135083 Otherwise:

135084 c. Call *open*( ) with O_CREAT|O_EXCL. The redirection succeeds or fails depending
135085 on whether the open succeeds or fails.

135086 2. The ``exclusive create first'' method.

135087 a. Call *open*( ) with O_CREAT|O_EXCL. If the open succeeds, the redirection
135088 succeeds. If the open fails with [EMFILE] or [ENFILE], use *stat*( ) to check whether
135089 a regular file exists; if it does, fail the redirection. Otherwise:

135090 b. Call *open*( ) without O_CREAT or O_TRUNC to open an existing file. If the open
135091 succeeds, use *fstat*( ) to check whether the opened file is a regular file. If it is, close
135092 it and fail the redirection. If it is a non-regular file, the redirection succeeds. If the
135093 second open fails, the redirection fails with a diagnostic based on the *errno* value
135094 set by the first open.

135095 (A minor variation of this method could also be used whereby step 2.b is only done if the
135096 *open*( ) in step 2.a fails with [EEXIST].)

135097 Method 1 is in widespread use. Method 2 has not been observed exactly as described, although
135098 an implementation which omits the *stat*( ) in step 2.a has been observed. Without the *stat*( ), this
135099 method has a problem in that if a regular file exists but the *open*( ) fails with [EMFILE] or
135100 [ENFILE] instead of [EEXIST] (which is to be expected if those conditions exist, because
135101 detecting [EEXIST] is more expensive), then the shell will give an incorrect diagnostic.
135102 (Reporting that no file descriptors are available implies that a non-regular file exists, because the
135103 shell tried to open the file and it is not supposed to open an existing regular file.)

135104 A variant of method 1 which omits the initial *stat*( ) call has also been observed; this has the
135105 same problem with [EMFILE] and [ENFILE]. With the *stat*( ), this misleading diagnostic can also
135106 happen, but only if a regular file is created in the timing window between steps 1.a and 1.b,
135107 which makes it an allowed case. (The standard allows a misleading diagnostic when there is
135108 concurrent creation or removal of files.)

135109 Both methods have cases where a misleading diagnostic is given when a non-regular file is
135110 concurrently created or removed. With method 1 it occurs if no file exists at steps 1.a and 1.b,
135111 and a non-regular file is created before step 1.c. With method 2 it occurs if a non-regular file
135112 exists at step 2.a and is removed before step 2.b. (In both cases, the diagnostic misleadingly
135113 implies that a regular file exists).

135114 Both methods differ from historical shell behavior in that the redirection fails if there is an
135115 existing symbolic link whose target does not exist, instead of the link's target being created as a
135116 regular file. The standard developers consider reliable lock file creation to be more important
135117 than the creation of symbolic link targets.

135118 Creation of lock files and unique (often temporary) files with *noclobber* set is only reliable
135119 provided neither non-regular files nor symbolic links to non-regular files exist or are created in
135120 the same directory with the same names, and no other processes delete the files while still in use.

If a directory such as **/tmp** is used for lock files, then another process could accidentally or maliciously create a FIFO (or a special file, given sufficient privilege) with the same name, causing multiple processes to simultaneously open the same lock file instead of one succeeding and the others failing.

Austin Group Defects 1016 and 1364 are applied, changing the requirements when the *noclobber* option is set.

C.2.7.3 *Appending Redirected Output*

Note that when a file is opened (even with the O_APPEND flag set), the initial file offset for that file is set to the beginning of the file. Some historic shells set the file offset to the current end-of-file when **append** mode shell redirection was used, but this is not allowed by POSIX.1-2024.

Austin Group Defect 1016 is applied, changing ``with the O_APPEND flag'' to ``with the O_APPEND flag set''.

C.2.7.4 *Here-Document*

Historical shell behavior was to treat the end of input as being equivalent to the delimiter of a here-document, terminating the here-document, usually without any indication, and continuing as if the delimiter had been recognized. This can cause problems where the delimiter had been intended to occur much earlier in the script, but was incorrectly entered—a mistake which for many other errors would have resulted in a syntax error, and an aborted script, instead simply generates incorrect results. Because of this some shell implementations have changed to reporting an undelimited here-document as a syntax error. Other implementations are encouraged to do the same.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0017 [890], XCU/TC2-2008/0018 [583], and XCU/TC2-2008/0019 [580] are applied.

Austin Group Defect 1036 is applied, clarifying how here-documents are parsed.

Austin Group Defect 1411 is applied, adding a paragraph break.

C.2.7.5 *Duplicating an Input File Descriptor*

The file descriptor duplication redirection operators, **[***n***]**<&*word* and **[***n***]**>&*word*, make a copy of one file descriptor as another. If the operation is successful, the new file descriptor has the same access mode as the source (old) file descriptor, because the access mode is determined by the open file description to which both file descriptors point. To avoid a redirection error, applications need to ensure that they use the appropriate redirection operator for the access mode of the file descriptor being duplicated.

Austin Group Defect 1536 is applied, making it optional whether attempting to duplicate an open file descriptor that is not open for input results in a redirection error.

C.2.7.6 *Duplicating an Output File Descriptor*

See Section C.2.7.5.

Austin Group Defect 1536 is applied, making it optional whether attempting to duplicate an open file descriptor that is not open for output results in a redirection error.

135159 *C.2.7.7 Open File Descriptors for Reading and Writing*

135160 There is no additional rationale provided for this section.

135161 C.2.8 Exit Status and Errors

135162 There is no additional rationale provided for this section.

135163 *C.2.8.1 Consequences of Shell Errors*

135164 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0020 [882] and XCU/TC2-2008/0021
135165 [717,882] are applied.

135166 Austin Group Defect 914 is applied, requiring that the shell does not exit when a redirection
135167 error occurs with compound commands or with function execution.

135168 Austin Group Defect 1427 is applied, changing this section to account for the effect of the
135169 *command* utility when it is used to execute a special built-in utility.

135170 Austin Group Defect 1629 is applied, requiring that the shell exits if an unrecoverable read error
135171 occurs when reading commands.

135172 *C.2.8.2 Exit Status for Commands*

135173 There is a historical difference in *sh* and *ksh* non-interactive error behavior. When a command
135174 named in a script is not found, some implementations of *sh* exit immediately, but *ksh* continues
135175 with the next command. Thus, the Shell and Utilities volume of POSIX.1-2024 says that the shell
135176 ``may'' exit in this case. This puts a small burden on the programmer, who has to test for
135177 successful completion following a command if it is important that the next command not be
135178 executed if the previous command was not found. If it is important for the command to have
135179 been found, it was probably also important for it to complete successfully. The test for successful
135180 completion would not need to change.

135181 Historically, shells have returned an exit status of 128+*n*, where *n* represents the signal number.
135182 Since signal numbers are not standardized, there is no portable way to determine which signal
135183 caused the termination. Also, it is possible for a command to exit with a status in the same range
135184 of numbers that the shell would use to report that the command was terminated by a signal.
135185 Implementations are encouraged to choose exit values greater than 256 to indicate programs that
135186 terminate by a signal so that the exit status cannot be confused with an exit status generated by a
135187 normal termination. However, the use of exit values greater than 256 poses a problem for the
135188 shell's own exit status. Historically this was the exit status of the last command invoked by the
135189 shell, but if the last command was terminated by a signal and was assigned an exit status greater
135190 than 256 by the shell, this value would be truncated to eight bits in the shell's exit status.
135191 Likewise truncation would occur with use of

135192
```
exit $?
```

135193 or

135194
```
ret=$?
135195 ....
135196 exit $ret
```

135197 in shell scripts. To avoid this truncation, shells which assign exit statuses greater than 256 are
135198 required to propagate the wait status of the last command to the shell's own wait status (by
135199 sending itself the same signal), and to handle exit values greater than 256 passed to the *exit*

135200 builtin by mimicking the wait status that would give rise to assignment of that exit status in the
135201 shell. Note that this requirement does not apply to signals that do not cause termination, such as
135202 SIGCHLD, since the shell can never actually assign a corresponding exit status greater than 256,
135203 and the requirement is worded in terms of this assignment.

135204 Historical shells make the distinction between ``utility not found'' and ``utility found but cannot
135205 execute'' in their error messages. By specifying two seldomly used exit status values for these
135206 cases, 127 and 126 respectively, this gives an application the opportunity to make use of this
135207 distinction without having to parse an error message that would probably change from locale to
135208 locale. The *command*, *env*, *nohup*, and *xargs* utilities in the Shell and Utilities volume of
135209 POSIX.1-2024 have also been specified to use this convention.

135210 When a command fails during word expansion or redirection, most historical implementations
135211 exit with a status of 1. However, there was some sentiment that this value should probably be
135212 much higher so that an application could distinguish this case from the more normal exit status
135213 values. Thus, the language ``greater than zero'' was selected to allow either method to be
135214 implemented.

135215 If a C application calls `exit(256)`, the command's exit status in the shell becomes zero due to
135216 the modulo 256 operation. Since zero is interpreted as ``true'' or ``success'' for **if** statements,
135217 AND and OR lists, `set -e`, and so on, applications should be careful to avoid exiting with a
135218 value that is a multiple of 256 unless the value is intended to be interpreted as true or success.

135219 To avoid ambiguity caused by the modulo 256 operation, applications are encouraged to avoid
135220 using a count or the result of a computation as the exit value unless the value is guaranteed to be
135221 non-negative and less than 256.

135222 The ambiguity caused by the modulo 256 operation is unfortunate, but required due to historical
135223 implementation behavior. A future version of this standard may change the definition of exit
135224 status to remove the modulo 256 requirement and use all bits of the value passed to *exit*() (or
135225 equivalent), and may introduce a way to select whether the special parameter `'?'` contains the
135226 exit status modulo 256 or the full exit status.

135227 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0022 [717] is applied.

135228 Austin Group Defect 51 is applied, clarifying the exit status when a command is terminated due
135229 to the receipt of a signal.

135230 Austin Group Defect 947 is applied, clarifying the exit status of commands.

135231 C.2.9 Shell Commands

135232 A description of an ``empty command'' was removed from an early proposal because it is only
135233 relevant in the cases of *sh* **−c ""**, *system*(`""`), or an empty shell-script file (such as the
135234 implementation of *true* on some historical systems). Since it is no longer mentioned in the Shell
135235 and Utilities volume of POSIX.1-2024, it falls into the silently unspecified category of behavior
135236 where implementations can continue to operate as they have historically, but conforming
135237 applications do not construct empty commands. (However, note that *sh* does explicitly state an
135238 exit status for an empty string or file.) In an interactive session or a script with other commands,
135239 extra <newline> or <semicolon> characters, such as:

```
135240 $ false
135241 $
135242 $ echo $?
135243 1
```

135244 would not qualify as the empty command described here because they would be consumed by

other parts of the grammar.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0023 [473] is applied.

C.2.9.1 Simple Commands

Austin Group Defect 1224 is applied, correcting a mismatch between the description of simple commands and the formal *simple_command* grammar.

Austin Group Defect 1227 is applied, inserting additional subsection headings.

Order of Processing

The enumerated list is used only when the command is actually going to be executed. For example, in:

```
true || $foo *
```

no expansions are performed.

Expansion of words in an assignment context following the command name can only occur for declaration utilities, and only when the word can be used as a variable assignment in isolation.

For example, this code sequence exports the single variable *a* with the value `"1 b=2"`, but invokes *make* with the macro *a* set to `'1'` and *b* set to `'2'`, since *make* is not a declaration utility:

```
set '1 b=2'
export a=$1
make a=$1
```

Conversely, this code sequence exports two variables, *a* set to `'1'` and *b* set to `'2'`, because the use of quoting means that the word could not be recognized as a variable assignment, and regular expansion rules require that field splitting occurs on the unquoted expansion of `$1`:

```
set '1 b=2'
export \a=$1
```

Likewise, this code sequence will not be parsed in assignment context, but is still required to export the variable named *foo* with the value `'1'`:

```
var=foo
export $var=1
```

Implementations are permitted to provide extensions that serve as declaration utilities, such as *typeset* or *local*, or even a way to define a function that can behave as a declaration utility.

Declaration utilities are only required to be recognized via lexical analysis; if any expansions are required before the command name is known, or before the first argument to the *command* utility is known, then it is unspecified whether subsequent arguments will be treated with an assignment context during expansion. For example, it is unspecified whether

```
var=export; $var a=~
```

sets the variable *a* to a literal <tilde> or to the value of `$HOME`, since lexical analysis sees `"$var"` rather than `"export"` as the command name.

Austin Group Defects 351 and 1535 are applied, adding requirements relating to declaration utilities.

135283 **Variable Assignments**

135284 The following example illustrates both how a variable assignment without a command name
135285 affects the current execution environment, and how an assignment with a command name only
135286 affects the execution environment of the command:

```
135287 $ x=red
135288 $ echo $x
135289 red
135290 $ export x
135291 $ sh -c 'echo $x'
135292 red
135293 $ x=blue sh -c 'echo $x'
135294 blue
135295 $ echo $x
135296 red
```

135297 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0021 [255] is applied.

135298 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0024 [654] is applied.

135299 Austin Group Defect 1009 is applied, clarifying the behavior when a special built-in utility is
135300 executed with a variable assignment.

135301 **Commands with no Command Name**

135302 This next example illustrates that redirections without a command name are still performed:

```
135303 $ ls foo
135304 ls: foo: no such file or directory
135305 $ > foo
135306 $ ls foo
135307 foo
```

135308 A command without a command name, but one that includes a command substitution, has an
135309 exit status of the last command substitution that the shell performed. For example:

```
135310 if      x=$(command)
135311 then    ...
135312 fi
```

135313 An example of redirections without a command name being performed in a subshell shows that
135314 the here-document does not disrupt the standard input of the **while** loop:

```
135315 IFS=:
135316 while    read a b
135317 do       echo $a
135318          <<-eof
135319          Hello
135320          eof
135321 done </etc/passwd
```

135322 Following are examples of commands without command names in AND-OR lists:

```
135323 > foo || {
135324     echo "error: foo cannot be created" >&2
135325     exit 1
135326 }

135327 # set saved if /vmunix.save exists
```



```
test -f /vmunix.save && saved=1
```

Command substitution and redirections without command names both occur in subshells, but they are not necessarily the same ones. For example, in:

```
exec 3> file
var=$(echo foo >&3) 3>&1
```

it is unspecified whether **foo** is echoed to the file or to standard output.

Austin Group Defect 1150 is applied, clarifying the exit status of a command that has no command name and has more than one command substitution.

Command Search and Execution

This description requires that the shell can execute shell scripts directly, even if the underlying system does not support the common `"#!"` interpreter convention. That is, if file **foo** contains shell commands and is executable, the following executes **foo**:

```
./foo
```

The command search shown here does not match all historical implementations. A more typical sequence has been:

- Any built-in (special or regular)
- Functions
- Path search for executable files

But there are problems with this sequence. Since the programmer has no idea in advance which utilities might have been built into the shell, a function cannot be used to override portably a utility of the same name. (For example, a function named *cd* cannot be written for many historical systems.) Furthermore, the *PATH* variable is partially ineffective in this case, and only a pathname with a <slash> can be used to ensure a specific executable file is invoked.

After the *execve*() failure described, the shell normally executes the file as a shell script. Some implementations, however, attempt to detect whether the file is actually a script and not an executable from some other architecture. The method used by the KornShell is allowed by the text that indicates non-text files may be bypassed.

The sequence selected for the Shell and Utilities volume of POSIX.1-2024 acknowledges that special built-ins cannot be overridden, but gives the programmer full control over which versions of other utilities are executed (with some exceptions). It provides a means of suppressing function lookup (via the *command* utility) for the user's own functions and, with the exception of the intrinsic utilities (see XCU Section 1.7, on page 2470), ensures that any regular built-ins or functions provided by the implementation are under the control of the path search. The mechanisms for associating non-intrinsic built-ins or functions with executable files in the path are not specified by the Shell and Utilities volume of POSIX.1-2024, but the wording requires that if either is implemented, the application is not able to distinguish a function or built-in from an executable (other than in terms of performance, presumably). The implementation ensures that all effects specified by the Shell and Utilities volume of POSIX.1-2024 resulting from the invocation of the regular built-in or function (interaction with the environment, variables, traps, and so on) are identical to those resulting from the invocation of an executable file.

Various historical implementations have used the names in item 1.b. as built-ins or reserved words. This standard does not specify their behavior, but their existence means that it is important for portable applications to avoid giving functions (or utilities in *PATH*) those names because the function (or utility in *PATH*) might not be executed as expected.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/4 is applied, updating the case where *execve*( ) fails due to an error equivalent to the [ENOEXEC] error.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0022 [168], XCU/TC1-2008/0023 [168], XCU/TC1-2008/0024 [168], XCU/TC1-2008/0025 [168], XCU/TC1-2008/0026 [168,430], XCU/TC1-2008/0027 [168,430], and XCU/TC1-2008/0028 [173] are applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0025 [935] and XCU/TC2-2008/0026 [705] are applied.

Austin Group Defect 465 is applied, adding *compound*, *enum*, *float*, *integer*, and *nameref* to the table of command names for which the results are unspecified.

Austin Group Defect 854 is applied, adding intrinsic utilities.

Austin Group Defect 1391 is applied, clarifying the execution of a standard utility provided by the implementation in the form of a function.

Standard File Descriptors

There is no additional rationale provided for this section.

Non-built-in Utility Execution

Austin Group Defect 1157 is applied, clarifying the execution of non-built-in utilities.

Austin Group Defects 1226 and 1435 are applied, clarifying the circumstances under which the shell may bypass execution of a non-built-in utility as a shell script.

Examples

Consider three versions of the *ls* utility:

1. The application includes a shell function named *ls*.
2. The user writes a utility named *ls* and puts it in **/fred/bin**.
3. The example implementation provides *ls* as a regular shell built-in that is invoked (either by the shell or directly by *exec*) when the path search reaches the directory **/posix/bin**.

If *PATH*=**/posix/bin**, various invocations yield different versions of *ls*:

Invocation	**Version of** *ls*
ls (from within application script)	(1) function
command ls (from within application script)	(3) built-in
ls (from within makefile called by application)	(3) built-in
system("*ls*")	(3) built-in
PATH="**/fred/bin**:$*PATH*" *ls*	(2) user's version

C.2.9.2 Pipelines

Because pipeline assignment of standard input or standard output or both takes place before redirection, it can be modified by redirection. For example:

```
$ command1 2>&1 | command2
```

sends both the standard output and standard error of *command1* to the standard input of *command2*.

The reserved word **!** allows more flexible testing using AND and OR lists. The behavior of **!(** is

unspecified because in the Korn Shell this introduces a negated pathname expansion. Portable applications need to separate the **!** and **(** to ensure the command is treated as a negated subshell.

It was suggested that it would be better to return a non-zero value if any command in the pipeline terminates with non-zero status (perhaps the bitwise-inclusive OR of all return values). However, the choice of the last-specified command semantics are historical practice and would cause applications to break if changed. An example of historical behavior:

```
$ sleep 5 | (exit 4)
$ echo $?
4
$ (exit 4) | sleep 5
$ echo $?
0
```

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0029 [205] is applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0027 [521] is applied.

Exit Status

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0030 [52] is applied.

Austin Group Defect 789 is applied, adding the *pipefail* option.

C.2.9.3 Lists

The equal precedence of "`&&`" and "`||`" is historical practice. The standard developers evaluated the model used more frequently in high-level programming languages, such as C, to allow the shell logical operators to be used for complex expressions in an unambiguous way, but they could not allow historical scripts to break in the subtle way unequal precedence might cause. Some arguments were posed concerning the "`{}`" or "`()`" groupings that are required historically. There are some disadvantages to these groupings:

- The "`()`" can be expensive, as they spawn other processes on some implementations. This performance concern is primarily an implementation issue.
- The "`{ }`" braces are not operators (they are reserved words) and require a trailing <space> after each '`{`', and a <semicolon> before each '`}`'. Most programmers (and certainly interactive users) have avoided braces as grouping constructs because of the problematic syntax required. Braces were not changed to operators because that would generate compatibility issues even greater than the precedence question; braces appear outside the context of a keyword in many shell scripts.

IEEE PASC Interpretation 1003.2 #204 is applied, clarifying that the operators "`&&`" and "`||`" are evaluated with left associativity.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0031 [45] and XCU/TC1-2008/0032 [45] are applied.

Asynchronous AND-OR Lists

Unless the implementation has an internal limit, such as {CHILD_MAX}, on the retained process IDs, it would require unbounded memory for the following example:

```
while true
do      foo & echo $!
done
```

The treatment of the signals SIGINT and SIGQUIT with asynchronous AND-OR lists is described in XCU Section 2.12 (on page 2521).

Since the connection of the input to the equivalent of **/dev/null** is considered to occur before redirections, the following script would produce no output:

```
exec < /etc/passwd
cat <&0 &
wait
```

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0028 [760] is applied.

Austin Group Defect 1254 is applied, replacing the *Asynchronous Lists* section with an *Asynchronous AND-OR Lists* section.

Sequential AND-OR Lists

Austin Group Defect 1254 is applied, replacing the *Sequential Lists* section with a *Sequential AND-OR Lists* section.

AND Lists

There is no additional rationale provided for this section.

OR Lists

There is no additional rationale provided for this section.

C.2.9.4 Compound Commands

Austin Group Defect 1309 is applied, clarifying the exit status of the **for**, **case**, **if**, **while**, and **until** compound commands.

Grouping Commands

The semicolon shown in {*compound-list*;} is an example of a control operator delimiting the } reserved word. Other delimiters are possible, as shown in XCU Section 2.10 (on page 2512); <newline> is frequently used.

A proposal was made to use the **<do-done>** construct in all cases where command grouping in the current process environment is performed, identifying it as a construct for the grouping commands, as well as for shell functions. This was not included because the shell already has a grouping construct for this purpose (`"{}"`), and changing it would have been counter-productive.

The requirement for conforming applications to separate two leading `'('` characters with white space if a grouping command would be parsed as an arithmetic expansion if preceded by a `'$'` is to allow shells which implement the `"(( arithmetic expression ))"` extension to apply the same disambiguation rules consistently to `$((...))` and `((...))`. See Section C.2.6.3 (on page 3884).

135487 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0033 [217] is applied.

135488 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0029 [473] is applied.

135489 **For Loop**

135490 The format is shown with generous usage of <newline> characters. See the grammar in XCU
135491 Section 2.10 (on page 2512) for a precise description of where <newline> and <semicolon>
135492 characters can be interchanged.

135493 Some historical implementations support `'{'` and `'}'` as substitutes for **do** and **done**. The
135494 standard developers chose to omit them, even as an obsolescent feature. (Note that these
135495 substitutes were only for the **for** command; the **while** and **until** commands could not use them
135496 historically because they are followed by compound-lists that may contain `"{...}"` grouping
135497 commands themselves.)

135498 The reserved word pair **do** ... **done** was selected rather than **do** ... **od** (which would have
135499 matched the spirit of **if** ... **fi** and **case** ... **esac**) because *od* is already the name of a standard
135500 utility.

135501 PASC Interpretation 1003.2 #169 has been applied changing the grammar.

135502 **Case Conditional Construct**

135503 An optional <left-parenthesis> before *pattern* was added to allow numerous historical KornShell
135504 scripts to conform. At one time, using the leading parenthesis was required if the **case** statement
135505 was to be embedded within a `"$()"` command substitution; this is no longer the case with the
135506 POSIX shell. Nevertheless, many historical scripts use the <left-parenthesis>, if only because it
135507 makes matching-parenthesis searching easier in *vi* and other editors. This is a relatively simple
135508 implementation change that is upwards-compatible for all scripts.

135509 Consideration was given to requiring *break* inside the *compound-list* to prevent falling through to
135510 the next pattern action list. This was rejected as being nonexisting practice. Instead, the standard
135511 now requires a feature first added in KornShell that using `";&"` instead of `";;"` as a terminator
135512 causes the exact opposite behavior—the flow of control continues with the next *compound-list*.

135513 Although the standard is explicit that the order of side-effects due to pattern expansion within a
135514 single clause is unspecified, it is clear that patterns are expanded in clause order, and that no
135515 further pattern expansions are attempted after the first match. That is, the following example is
135516 required to output `"1.0"`:

```
135517 x=0 y=1
135518 case 1 in
135519   $((y=0)) ) ;;
135520   $((x=1)) ) ;&
135521   $((x=2)) ) echo $x.$y ;;
135522 esac
```

135523 Some implementations of the shell also allow `";;&"` as a terminator which falls through to the
135524 next matching pattern (regardless of the choice of terminator in any intermediate non-matching
135525 clauses), in contrast to `";&"` falling through to the next clause (regardless of the pattern
135526 guarding that clause). This is an allowed extension, but is not required by the standard at this
135527 time.

135528 The pattern `'*'`, given as the last pattern in a **case** construct, is equivalent to the default case in
135529 a C-language **switch** statement.

135530 The grammar shows that reserved words can be used as patterns, even if one is the first word on
135531 a line. Obviously, the reserved word **esac** cannot be used in this manner.

Some historical shells would fall back to doing a byte to byte comparison with each pattern if the pattern matching rules did not produce a match. That behavior is not allowed by this standard because it allows user input to bypass input validations like:

```
case $1 in
  [0123456789]) : OK;;
  *) echo >&2 not a decimal digit; exit 1;;
esac
```

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0029 [473] is applied.

Austin Group Defect 449 is applied, adding `;&` as a **case** clause terminator.

Austin Group Defect 1454 is applied, clarifying that a **case** statement with no patterns is valid syntax.

If Conditional Construct

The precise format for the command syntax is described in XCU Section 2.10 (on page 2512).

While Loop

The precise format for the command syntax is described in XCU Section 2.10 (on page 2512).

Until Loop

The precise format for the command syntax is described in XCU Section 2.10 (on page 2512).

C.2.9.5 *Function Definition Command*

The description of functions in an early proposal was based on the notion that functions should behave like miniature shell scripts; that is, except for sharing variables, most elements of an execution environment should behave as if they were a new execution environment, and changes to these should be local to the function. For example, traps and options should be reset on entry to the function, and any changes to them do not affect the traps or options of the caller. There were numerous objections to this basic idea, and the opponents asserted that functions were intended to be a convenient mechanism for grouping common commands that were to be executed in the current execution environment, similar to the execution of the *dot* special built−in.

It was also pointed out that the functions described in that early proposal did not provide a local scope for everything a new shell script would, such as the current working directory, or *umask*, but instead provided a local scope for only a few select properties. The basic argument was that if a local scope is needed for the execution environment, the mechanism already existed: the application can put the commands in a new shell script and call that script. All historical shells that implemented functions, other than the KornShell, have implemented functions that operate in the current execution environment. Because of this, traps and options have a global scope within a shell script. Local variables within a function were considered and included in another early proposal (controlled by the special built-in *local*), but were removed because they do not fit the simple model developed for functions and because there was some opposition to adding yet another new special built-in that was not part of historical practice. Implementations should reserve the identifier *local* (as well as *typeset*, as used in the KornShell) in case this local variable mechanism is adopted in a future version of this standard.

A separate issue from the execution environment of a function is the availability of that function to child shells. A few objectors maintained that just as a variable can be shared with child shells by exporting it, so should a function. In early proposals, the *export* command therefore had a **−f**

135575 flag for exporting functions. Functions that were exported were to be put into the environment
135576 as *name*( )=*value* pairs, and upon invocation, the shell would scan the environment for these and
135577 automatically define these functions. This facility was strongly opposed and was omitted. Some
135578 of the arguments against exportable functions were as follows:

135579 • There was little historical practice. The Ninth Edition shell provided them, but there was
135580 controversy over how well it worked.

135581 • There are numerous security problems associated with functions appearing in the
135582 environment of a user and overriding standard utilities or the utilities owned by the
135583 application.

135584 • There was controversy over requiring *make* to import functions, where it has historically
135585 used an *exec* function for many of its command line executions.

135586 • Functions can be big and the environment is of a limited size. (The counter-argument was
135587 that functions are no different from variables in terms of size: there can be big ones, and
135588 there can be small ones—and just as one does not export huge variables, one does not
135589 export huge functions. However, this might not apply to the average shell-function writer,
135590 who typically writes much larger functions than variables.)

135591 As far as can be determined, the functions in the Shell and Utilities volume of POSIX.1-2024
135592 match those in System V. Earlier versions of the KornShell had two methods of defining
135593 functions:

135594 `function` *`fname`* `{` *`compound-list`* `}`

135595 and:

135596 *`fname`*`() {` *`compound-list`* `}`

135597 The latter used the same definition as the Shell and Utilities volume of POSIX.1-2024, but
135598 differed in semantics, as described previously. The current edition of the KornShell aligns the
135599 latter syntax with the Shell and Utilities volume of POSIX.1-2024 and keeps the former as is.

135600 Some shells accept simple commands (see XCU Section 2.9.1, on page 2500) after *`fname`*`()` in
135601 addition to compound commands (see XCU Section 2.9.4, on page 2508); however this standard
135602 only requires support for compound commands.

135603 The name space for functions is limited to that of a *name* because of historical practice.
135604 Complications in defining the syntactic rules for the function definition command and in
135605 dealing with known extensions such as the `"@()"` usage in the KornShell prevented the name
135606 space from being widened to a *word*. Using functions to support synonyms such as the `"!!"`
135607 and `'%'` usage in the C shell is thus disallowed to conforming applications, but acceptable as an
135608 extension. For interactive users, the aliasing facilities in the Shell and Utilities volume of
135609 POSIX.1-2024 should be adequate for this purpose. It is recognized that the name space for
135610 utilities in the file system is wider than that currently supported for functions, if the portable
135611 filename character set guidelines are ignored, but it did not seem useful to mandate extensions
135612 in systems for so little benefit to conforming applications.

135613 The `"()"` in the function definition command consists of two operators. Therefore, intermixing
135614 <blank> characters with the *fname*, `'('`, and `')'` is allowed, but unnecessary.

135615 An example of how a function definition can be used wherever a simple command is allowed:

```
135616 # If variable i is equal to "yes",
135617 # define function foo to be ls -l
135618 #
135619 [ "$i" = yes ] && foo() {
135620     ls -l
```


135621 }

135622 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0034 [383] and XCU/TC1-2008/0035
135623 [214] are applied.

135624 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0029 [473] and XCU/TC2-2008/0030
135625 [654] are applied.

135626 C.2.10 Shell Grammar

135627 There are several subtle aspects of this grammar where conventional usage implies rules about
135628 the grammar that in fact are not true.

135629 For *compound_list*, only the forms that end in a *separator* allow a reserved word to be recognized,
135630 so usually only a *separator* can be used where a compound list precedes a reserved word (such as
135631 **Then**, **Else**, **Do**, and **Rbrace**). Explicitly requiring a separator would disallow such valid (if rare)
135632 statements as:

135633 `if (false) then (echo x) else (echo y) fi`

135634 See the Note under special grammar rule (1).

135635 Concerning the third sentence of rule (1) (``Also, if the parser ...''):

135636 • This sentence applies rather narrowly: when a compound list is terminated by some clear
135637 delimiter (such as the closing **fi** of an inner **if_clause**) then it would apply; where the
135638 compound list might continue (as in after a `';'`), rule (7a) (and consequently the first
135639 sentence of rule (1)) would apply. In many instances the two conditions are identical, but
135640 this part of rule (1) does not give license to treating a **WORD** as a reserved word unless it
135641 is in a place where a reserved word has to appear.

135642 • The statement is equivalent to requiring that when the LR(1) lookahead set contains
135643 exactly one reserved word, it must be recognized if it is present. (Here ``LR(1)'' refers to the
135644 theoretical concepts, not to any real parser generator.)

135645 For example, in the construct below, and when the parser is at the point marked with `'^'`,
135646 the only next legal token is **then** (this follows directly from the grammar rules):

```
135647 if if...fi then ... fi
135648            ^
```

135649 At that point, the **then** must be recognized as a reserved word.

135650 (Depending on the parser generator actually used, ``extra'' reserved words may be in some
135651 lookahead sets. It does not really matter if they are recognized, or even if any possible
135652 reserved word is recognized in that state, because if it is recognized and is not in the
135653 (theoretical) LR(1) lookahead set, an error is ultimately detected. In the example above, if
135654 some other reserved word (for example, **while**) is also recognized, an error occurs later.

135655 This is approximately equivalent to saying that reserved words are recognized after other
135656 reserved words (because it is after a reserved word that this condition occurs), but avoids
135657 the ``except for ...'' list that would be required for **case**, **for**, and so on. (Reserved words
135658 are of course recognized anywhere a *simple_command* can appear, as well. Other rules take
135659 care of the special cases of non-recognition, such as rule (4) for **case** statements.)

135660 Note that the body of here-documents are handled by token recognition (see XCU Section 2.3, on
135661 page 2475) and do not appear in the grammar directly. (However, the here-document I/O
135662 redirection operator is handled as part of the grammar.)

135663 The optional redirection syntax:

135664 **{***location***}***redir-op word*

135665 (see XCU Section 2.7, on page 2493) is accommodated in the grammar rules by the optional
135666 **IO_LOCATION** token identifier and two correspondingly optional elements in `io_redirect`.
135667 Without these, the grammar would not permit this form of redirection because it would require
135668 that, for example, `echo {var}> foo` is parsed such that `{var}` is a **WORD** to be expanded
135669 and passed to *echo*. The grammar does not restrict the location given between the `'{'` and `'}'`
135670 in these forms (other than requiring it to be non-empty) since shells may parse an invalid
135671 location as part of an `io_redirect` and later treat the invalid location as an error.

135672 *C.2.10.1 Shell Grammar Lexical Conventions*

135673 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0031 [648] and XCU/TC2-2008/0032
135674 [574,646] are applied.

135675 Austin Group Defect 1193 is applied, adding the optional **IO_LOCATION** token identifier.

135676 Austin Group Defect 1454 is applied, clarifying how to convert the token identifier type of the
135677 **TOKEN** when rule 1 applies.

135678 *C.2.10.2 Shell Grammar Rules*

135679 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0036 [44] is applied.

135680 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0033 [643,839], XCU/TC2-2008/0034
135681 [643], XCU/TC2-2008/0035 [648], XCU/TC2-2008/0036 [736], XCU/TC2-2008/0037 [737],
135682 XCU/TC2-2008/0038 [581], and XCU/TC2-2008/0039 [735] are applied.

135683 Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

135684 Austin Group Defect 449 is applied, adding `;&` as a **case** clause terminator.

135685 Austin Group Defect 1193 is applied, adding the optional **IO_LOCATION** token identifier.

135686 Austin Group Defects 1276 and 1279 are applied, clarifying rule 7.

135687 Austin Group Defect 1454 is applied, clarifying how rule 4 applies.

135688 ## C.2.11 Job Control

135689 See also Job Control (on page 3656).

135690 Shell implementations differ regarding how much of a foreground job is retained when it is
135691 converted to a suspended job. For example, given this foreground job:

135692 ```
sleep 10; echo foo; echo bar &
```

135693 if this is suspended during execution of the sleep, *ksh93* retains all of the commands in the
135694 suspended job and executes them when *fg* is used:

135695 ```
^Z[1] + Stopped                    sleep 10; echo foo; echo bar &
135696 $ jobs
135697 [1] + Stopped                 sleep 10; echo foo; echo bar &
135698 $ fg
135699 sleep 10; echo foo; echo bar
135700 foo
135701 [1]     30686
135702 bar
```


135703 **$**

135704 However, some other shells create a suspended job containing only the `sleep 10` command.

135705 Some historical shells did not handle suspending a foreground AND-OR list well. They would
135706 treat the wait status of a process that indicated it had stopped as if it was a non-zero exit status
135707 and (if the next operator in the AND-OR list was `||`) would execute the remainder of the AND-
135708 OR list at that point. This behavior is not allowed by the standard for two reasons:

135709 1. It does not meet the fundamental requirement of an AND-OR list that the decision on
135710 whether to execute each part (except the first) is made based on the exit status of the
135711 previous part when it completes.

135712 2. It can lead to data loss. For example, consider a user who often runs this command:

135713 ```
generate_report > report.out || rm report.out
```

135714 with the intention that the incomplete results from a failed *generate_report* run are never
135715 retained in order that they cannot be mistaken for a complete set of results. If one day the
135716 user decides to check on the progress of the command by stopping it and examining what
135717 has been written so far, they will find that the **report.out** file has already been removed.

135718 Austin Group Defects 1254 and 1675 are applied, adding this section.

## 135719 C.2.12 Signals and Error Handling

135720 Historically, some shell implementations silently ignored attempts to use *trap* to set SIGINT or
135721 SIGQUIT to the default action or to set a trap for them after they have been set to be ignored by
135722 the shell when it executes an asynchronous subshell (and job control is disabled). This behavior
135723 is not conforming. For example, if a shell script containing the following line is run in the
135724 foreground at a terminal:

135725 ```
(trap - INT; exec sleep 10) & wait
```

135726 and is then terminated by typing the interrupt character, this standard requires that the *sleep*
135727 command is terminated by the SIGINT signal.

135728 SD5-XCU-ERN-93 is applied, updating the first paragraph of XCU Section 2.12 (on page 2521).

135729 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0040 [750] is applied.

135730 C.2.13 Shell Execution Environment

135731 Some implementations have implemented the last stage of a pipeline in the current environment
135732 so that commands such as:

135733 ```
command | read foo
```

135734 set variable **foo** in the current environment. This extension is allowed, but not required;
135735 therefore, a shell programmer should consider a pipeline to be in a subshell environment, but
135736 not depend on it.

135737 In early proposals, the description of execution environment failed to mention that each
135738 command in a multiple command pipeline could be in a subshell execution environment. For
135739 compatibility with some historical shells, the wording was phrased to allow an implementation
135740 to place any or all commands of a pipeline in the current environment. However, this means that
135741 a POSIX application must assume each command is in a subshell environment, but not depend
135742 on it.

  
```

The wording about shell scripts is meant to convey the fact that describing ``trap actions'' can only be understood in the context of the shell command language. Outside of this context, such as in a C-language program, signals are the operative condition, not traps.

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0037 [238] is applied.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0041 [706] is applied.

Austin Group Defect 1247 is applied, changing ``signal traps'' to ``traps'' and changing ``All other commands'' to ``Except where otherwise stated, all other commands''.

Austin Group Defect 1254 is applied, changing the list item relating to process IDs ``known to this shell environment''.

Austin Group Defect 1384 is applied, changing the requirements for subshells of interactive shells.

Austin Group Defect 1580 is applied, adding a list item about environment variables with invalid names.

C.2.14 Pattern Matching Notation

Pattern matching is a simpler concept and has a simpler syntax than REs, as the former is generally used for the manipulation of filenames, which are relatively simple collections of characters, while the latter is generally used to manipulate arbitrary text strings of potentially greater complexity. However, some of the basic concepts are the same, so this section points liberally to the detailed descriptions in XBD Chapter 9 (on page 179).

Austin Group Defect 1443 is applied, adding non-shell uses to the description of what shell pattern matching notation is used for.

Austin Group Defect 1564 is applied, clarifying that pattern matching notation is used for matching character strings (not arbitrary byte strings), and that if an attempt is made to use pattern matching notation to match a string that contains one or more bytes that do not form part of a valid character, the behavior is unspecified.

C.2.14.1 *Patterns Matching a Single Character*

Both quoting and escaping are described here because pattern matching must work in three separate circumstances:

1. Calling directly upon the shell, such as in pathname expansion or in a **case** statement. All of the following match the string or file **abc**:

   ```
   abc "abc" a"b"c a\bc a[b]c a["b"]c a[\b]c a["\b"]c a?c a*c
   ```

   The following do not:

   ```
   "a?c" a\*c a\[b]c
   ```

2. Calling a utility or function without going through a shell, as described for *find* and the *fnmatch*() and *glob*() functions defined in the System Interfaces volume of POSIX.1-2024, or pattern matching in the shell in situations where the pattern is specified indirectly instead of directly to the shell, such as:

   ```
   ls -ld -- $pattern
   ```

   or

   ```
   case $var in ($pattern) ...
   ```


3. Calling utilities such as *find*, *cpio*, *tar*, or *pax* through the shell command line. In this case, shell quote removal is performed before the utility sees the argument. For example, in:

   ```
   find /bin -name "e\c[\h]o" -print
   ```

   after quote removal, the <backslash> characters are presented to *find* and it treats them as escape characters. Both precede ordinary characters, so the *c* and *h* represent themselves and *echo* would be found on many historical systems (that have it in **/bin**). To find a filename that contained shell special characters or pattern characters, both quoting and escaping are required, such as:

   ```
   pax -r ... "*a(\?"
   ```

   to extract a filename ending with `"a(?"`.

The wording ``In a pattern, or part of one, where a shell-quoting <backslash> cannot be used to preserve the literal value of a character that would otherwise be treated as special'' has been carefully crafted so that for the shell it only applies to certain contexts. In particular:

- The use of ``or part of one'' is needed because a single pattern can be produced partly from characters directly included in a word and partly from characters that result from one or more of the word expansions. For example, in the following command the <backslash> escapes the `'?'` character:

  ```
  dir='abc\?'
  ls -l -- $dir/*.c
  ```

- The reference to ``a shell-quoting <backslash>'' rather than just using ``where shell quoting cannot be used'' is because there are ways that other types of shell quoting can be used where a shell-quoting <backslash> cannot, such as placing an expansion within double-quotes as in this example:

  ```
  dir='abc?'
  ls -l -- "$dir"/*.c
  ```

- The use of ``that would otherwise be treated as special'' is needed because otherwise the condition would apply to <backslash> in single-quotes. For example, in the following command the <backslash> is not treated as escaping the `'?'` because the `'?'` would not be treated as special anyway:

  ```
  ls -l 'abc\?'/*.c
  ```

In patterns specified indirectly to the shell, it is unspecified whether or not <backslash> is special inside bracket expressions. This is because there are two mutually exclusive consistency aims and neither is considered more important than the other. One is consistency with direct patterns, where <backslash> is special inside bracket expressions (which is, in turn, for consistency with the way single-quotes and double-quotes preserve the literal value of characters inside bracket expressions); the other is consistency with regular expressions, *find*, *pax*, *fnmatch*(), and *glob*(), where <backslash> is not special inside bracket expressions (not counting the extra C-string escaping in EREs in *awk*).

Earlier versions of this standard allowed two behaviors when a pattern ends with an unescaped <backslash>: it could match nothing or be treated as an invalid pattern. However, a third behavior has since been observed, where the ending <backslash> is treated as a literal <backslash>, and therefore this standard now simply states that the behavior is unspecified.

Earlier versions of this standard included the statement ``The shell special characters always require quoting'' in XCU Section 2.14.1 (on page 2523). It is unclear what was intended by this, since there are pattern matching contexts in which it is not possible to quote those characters, such as:


```
execlp("find", "find", ".", "-name", "*[()]*", (char *)0);
```

where the parentheses cannot be escaped with a <backslash> because <backslash> is not special in bracket expressions in that context. The statement is thought to have been a warning to application writers and interactive shell users that shell special characters (sometimes called metacharacters) always need quoting in patterns that appear directly in shell code; for example, this code:

```
case $char in
[()]) ... ;;
esac
```

is incorrect because the parentheses are parsed as operators—they need to be quoted in order to be treated as part of the pattern. This standard now simply requires instead that applications quote or escape any character that would otherwise be treated as special, in order for it to be matched as an ordinary character. If shell special characters are used without this protection in contexts where they are treated as special, syntax errors can result or implementation extensions can be triggered. Some shells support a series of extensions based on parentheses in patterns that are valid extensions in these contexts because they would otherwise cause syntax errors. However, this means that they are not allowed by this standard to be recognized in contexts where those syntax errors would not occur anyway, such as in:

```
pattern='a*(b)'; ls -- $pattern
```

which this standard requires to list files with names beginning `'a'` and ending `"(b)"`. It is recommended that implementations do not extend pattern matching in the shell in ways that are only valid extensions because they would otherwise be syntax errors, in order to avoid inconsistency between different pattern matching contexts. One way to provide an extension that is consistent between different pattern matching contexts in the shell (although still not consistent with *find –name*, *fnmatch*(), etc.) is to enable the extension only when a non-standard shell option is set, or when the shell is executed using a command name other than *sh*. Consistency with non-shell contexts can then be achieved by enabling equivalent extensions in those other contexts by use of non-standard utility options or non-standard FNM_* and GLOB_* flags.

The restriction on a <circumflex> in a bracket expression is to allow implementations that support pattern matching using the <circumflex> as the negation character in addition to the <exclamation-mark>. A conforming application must use something like `"[\^!]"` to match either character.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0042 [806] is applied.

Austin Group Defect 985 is applied, changing the description of the `'['` special character.

Austin Group Defect 1234 is applied, clarifying how <backslash> is handled in patterns.

C.2.14.2 *Patterns Matching Multiple Characters*

Since each <asterisk> matches zero or more occurrences, the patterns `"a*b"` and `"a**b"` have identical functionality.

Examples

`a[bc]`	Matches the strings `"ab"` and `"ac"`.
`a*d`	Matches the strings `"ad"`, `"abd"`, and `"abcd"`, but not the string `"abc"`.
`a*d*`	Matches the strings `"ad"`, `"abcd"`, `"abcdef"`, `"aaaad"`, and `"adddd"`.
`*a*d`	Matches the strings `"ad"`, `"abcd"`, `"efabcd"`, `"aaaad"`, and `"adddd"`.

C.2.14.3 *Patterns Used for Filename Expansion*

The caveat about a <slash> within a bracket expression is derived from historical practice. The pattern `"a[b/c]d"` does not match such pathnames as **abd** or **a/d**. On some implementations (including those conforming to the Single UNIX Specification), it matched a pathname of literally `"a[b/c]d"`. On other systems, it produced an undefined condition (an unescaped `'['` used outside a bracket expression). In this version, the XSI behavior is now required.

Filenames beginning with a <period> historically have been specially protected from view on UNIX systems. A proposal to allow an explicit <period> in a bracket expression to match a leading <period> was considered; it is allowed as an implementation extension, but a conforming application cannot make use of it. If this extension becomes popular in the future, it will be considered for a future version of the Shell and Utilities volume of POSIX.1-2024.

Patterns are matched against existing filenames and pathnames only when the pattern contains a `'*'`, `'?'` or `'['` character that will be treated as special. This prevents accidental removal of <backslash> characters in variable expansions where generating a list of matching files is not intended and a (usually oddly named) file with a matching name happens to exist. For example, a shell script that tries to be portable to systems that predate the introduction of functions and *printf* might use this on POSIX systems:

```
myecho='printf %s\n'
```

to be used as:

```
$myecho args...
```

If `%s\n` were to be matched against existing files, this would not work if a file called **%sn** happened to exist.

Historical systems have varied in their permissions requirements. To match **f*/bar** has required read permissions on the **f*** directories in the System V shell, but the Shell and Utilities volume of POSIX.1-2024, the C shell, and KornShell require only search permissions. If read or search permission is denied, shells do not report an error but treat this as a successful ``no match'' condition. Error conditions that are related to file system contents and occur when attempting to read or search a directory are also required to be treated the same way because they imply that there are no matches (that are accessible to the process). For example, if the pattern is **foo/*bar** and attempting to open the directory **foo** fails because it does not exist or is not a directory, then there can be no matching pathnames. The error conditions listed in XSH Section 2.3 (on page 507) that are related to file system contents and could occur when attempting to open or search a directory are [EACCES], [ELOOP], [ENAMETOOLONG], [ENOENT], and [ENOTDIR]. Error conditions that are not related to file system contents or which occur when reading a directory, notably [EMFILE] and [ENFILE] but also things like [EIO], [ENOMEM], and [EOVERFLOW], can either be treated as errors or be treated the same way as when permission is denied. Treating them as errors is seen as desirable, because to do otherwise would mean the shell could execute a command with an unchanged pattern when pathnames matching the pattern exist, but it is not historical practice. Implementations that handle the two categories of error differently should also handle non-standard error conditions appropriately, if encountered, depending on which

category they fit into.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0043 [963] is applied.

Austin Group Defect 1070 is applied, requiring that when the matching filenames or pathnames are sorted, any that collate equally are further compared byte-by-byte using the collating sequence for the POSIX locale.

Austin Group Defect 1228 is applied, allowing directory entries for dot and dot-dot to be ignored when matching patterns against existing filenames.

Austin Group Defect 1234 is applied, changing the behavior of patterns used for filename expansion such that a pattern is matched against existing filenames and pathnames only when it contains a `'*'`, `'?'` or `'['` character that will be treated as special.

Austin Group Defects 1273 and 1275 are applied, clarifying how errors are treated when attempting to open or search a pathname as a directory or attempting to read an opened directory.

C.2.15 Special Built-In Utilities

See the RATIONALE sections on the individual reference pages.

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0044 [882] and XCU/TC2-2008/0045 [654] are applied.

Austin Group Defect 1009 is applied, clarifying the behavior when a special built-in utility is executed with a variable assignment.

Austin Group Defect 1445 is applied, changing text relating to the term ``built-in''.

C.3 Utilities

For the utilities included in POSIX.1-2024, see the RATIONALE sections on the individual reference pages.

C.3.1 Utilities Removed in this Version

The following utilities were removed in this version of this standard:

qalter	*qmove*	*qrls*	*qstat*
qdel	*qmsg*	*qselect*	*qsub*
qhold	*qrerun*	*qsig*	

C.3.2 Utilities Removed in the Previous Version

None.

C.3.3 Exclusion of Utilities

The set of utilities contained in POSIX.1-2024 is drawn from the base documents for IEEE Std 1003.2-1992, with one addition: the *c17* utility. This section contains rationale for some of the deliberations that led to this set of utilities, and why certain utilities were excluded.

Many utilities were evaluated by the standard developers; more historical utilities were excluded from the base documents for IEEE Std 1003.2-1992 than included. The following list contains many common UNIX system utilities that were not included as mandatory utilities, in the User Portability Utilities option, in the XSI option, or in one of the software development groups. It is logistically difficult for this rationale to distribute correctly the reasons for not including a utility among the various utility options. Therefore, this section covers the reasons for all utilities not included in POSIX.1-2024.

This rationale is limited to a discussion of only those utilities actively or indirectly evaluated by the IEEE Std 1003.2-1992 standard developers, rather than the list of all known UNIX utilities from all its variants.

adb The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool. Furthermore, many useful aspects of *adb* are very hardware-specific.

as Assemblers are hardware-specific and are included implicitly as part of the compilers in POSIX.1-2024.

banner The only known use of this command is as part of the *lp* printer header pages. It was decided that the format of the header is implementation-defined, so this utility is superfluous to application portability.

calendar This reminder service program is not useful to conforming applications.

cancel The *lp* (line printer spooling) system specified is the most basic possible and did not need this level of application control.

chroot This is primarily of administrative use, requiring superuser privileges.

col No utilities defined in POSIX.1-2024 produce output requiring such a filter. The *nroff* text formatter is present on many historical systems and will continue to remain as an extension; *col* is expected to be shipped by all the systems that ship *nroff*.

cpio This has been replaced by *pax*, for reasons explained in the rationale for that utility.

cpp This is subsumed by *c17*.

cu This utility is terminal-oriented and is not useful from shell scripts or typical application programs.

dc The functionality of this utility can be provided by the *bc* utility; *bc* was selected because it was easier to use and had superior functionality. Although the historical versions of *bc* are implemented using *dc* as a base, POSIX.1-2024 prescribes the interface and not the underlying mechanism used to implement it.

dircmp Although a useful concept, the historical output of this directory comparison program is not suitable for processing in application programs. Also, the *diff* **–r** command gives equivalent functionality.

dis Disassemblers are hardware-specific.

emacs The community of *emacs* editing enthusiasts was adamant that the full *emacs* editor not be included in IEEE Std 1003.2-1992 because they were concerned that an attempt to standardize this very powerful environment would encourage vendors to ship versions conforming strictly to the standard, but lacking the extensibility required by the community. The author of the original *emacs* program also expressed his desire to omit the program. Furthermore, there were a number of historical UNIX systems that did not include *emacs*, or included it without supporting it, but there were very few that did not include and support *vi*.

ld This is subsumed by *c17*.

line The functionality of *line* can be provided with *read*.

lint This technology is partially subsumed by *c17*. It is also hard to specify the degree of checking for possible error conditions in programs in any compiler, and specifying what *lint* would do in these cases is equally difficult.

It is fairly easy to specify what a compiler does. It requires specifying the language, what it does with that language, and stating that the interpretation of any incorrect program is unspecified. Unfortunately, any description of *lint* is required to specify what to do with erroneous programs. Since the number of possible errors and questionable programming practices is infinite, one cannot require *lint* to detect all errors of any given class.

Additionally, some vendors complained that since many compilers are distributed in a binary form without a *lint* facility (because the ISO C standard does not require one), implementing the standard as a stand-alone product will be much harder. Rather than being able to build upon a standard compiler component (simply by providing *c17* as an interface), source to that compiler would most likely need to be modified to provide the *lint* functionality. This was considered a major burden on system providers for a very small gain to developers (users).

login This utility is terminal-oriented and is not useful from shell scripts or typical application programs.

lorder This utility is an aid in creating an implementation-defined detail of object libraries that the standard developers did not feel required standardization.

lpstat The *lp* system specified is the most basic possible and did not need this level of application control.

mail This utility was omitted in favor of *mailx* because there was a considerable functionality overlap between the two.

mknod This was omitted in favor of *mkfifo*, as *mknod* has too many implementation-defined functions.

news This utility is terminal-oriented and is not useful from shell scripts or typical application programs.

pack This compression program was considered inferior to *compress*.

passwd This utility was proposed in an early draft of the IEEE Std 1003.2-1992 UPE but met with too many objections to be included. There were various reasons:

- Changing a password should not be viewed as a command, but as part of the login sequence. Changing a password should only be done while a trusted path is in effect.

136030 • Even though the text in early drafts was intended to allow a variety of
136031 implementations to conform, the security policy for one site may differ from
136032 another site running with identical hardware and software. One site might
136033 use password authentication while the other did not. Vendors could not
136034 supply a *passwd* utility that would conform to POSIX.1-2024 for all sites using
136035 their system.

136036 • This is really a subject for a system administration working group or a
136037 security working group.

136038 *pcat* This compression program was considered inferior to *zcat*.

136039 *pg* This duplicated many of the features of the *more* pager, which was preferred by the
136040 standard developers.

136041 *prof* The intent of the various software development utilities was to assist in the
136042 installation (rather than the actual development and debugging) of applications.
136043 This utility is primarily a debugging tool.

136044 RCS RCS was originally considered as part of a version control utilities portion of the
136045 scope. However, this aspect was abandoned by the standard developers. SCCS is
136046 now included as an optional part of the XSI option.

136047 *red* Restricted editor. This was not considered by the standard developers because it
136048 never provided the level of security restriction required.

136049 *rsh* Restricted shell. This was not considered by the standard developers because it
136050 does not provide the level of security restriction that is implied by historical
136051 documentation.

136052 *sdb* The intent of the various software development utilities was to assist in the
136053 installation (rather than the actual development and debugging) of applications.
136054 This utility is primarily a debugging tool. Furthermore, some useful aspects of *sdb*
136055 are very hardware-specific.

136056 *sdiff* The ``side-by-side *diff*'' utility from System V was omitted because it is used
136057 infrequently, and even less so by conforming applications. Despite being in
136058 System V, it is not in the SVID or XPG.

136059 *shar* Any of the numerous ``shell archivers'' were excluded because they did not meet
136060 the requirement of existing practice.

136061 *shl* This utility is terminal-oriented and is not useful from shell scripts or typical
136062 application programs. The job control aspects of the shell command language are
136063 generally more useful.

136064 *size* The intent of the various software development utilities was to assist in the
136065 installation (rather than the actual development and debugging) of applications.
136066 This utility is primarily a debugging tool.

136067 *spell* This utility is not useful from shell scripts or typical application programs. The
136068 *spell* utility was considered, but was omitted because there is no known technology
136069 that can be used to make it recognize general language for user-specified input
136070 without providing a complete dictionary along with the input file.

136071 *su* This utility is not useful from shell scripts or typical application programs. (There
136072 was also sentiment to avoid security-related utilities.)

136073	*sum*	This utility was renamed *cksum*.
136074	*tar*	This has been replaced by *pax*, for reasons explained in the rationale for that utility.
136075	*unpack*	This compression program was considered inferior to *uncompress*.
136076 136077	*wall*	This utility is terminal-oriented and is not useful in shell scripts or typical applications. It is generally used only by system administrators.

Rationale (Informative)

Part D:

Portability Considerations

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

136083 *Appendix D*

136084 # *Portability Considerations (Informative)*

136085 This section contains information to satisfy various international requirements:

136086 • Section D.1 describes perceived user requirements.

136087 • Section D.2 (on page 3923) indicates how the facilities of POSIX.1-2024 satisfy those
136088 requirements.

136089 • Section D.3 (on page 3931) offers guidance to writers of profiles on how the configurable
136090 options, limits, and optional behavior of POSIX.1-2024 should be cited in profiles.

136091 ## D.1 User Requirements

136092 This section describes the user requirements that were perceived by the standard developers.
136093 The primary source for these requirements was an analysis of historical practice in widespread
136094 use, as typified by the base documents for the ISO POSIX-1: 1996 standard.

136095 POSIX.1-2024 addresses the needs of users requiring open systems solutions for source code
136096 portability of applications. It currently addresses users requiring open systems solutions for
136097 source-code portability of applications involving multi-programming and process management
136098 (creating processes, signaling, and so on); access to files and directories in a hierarchy of file
136099 systems (opening, reading, writing, deleting files, and so on); access to asynchronous
136100 communications ports and other special devices; access to information about other users of the
136101 system; facilities supporting applications requiring bounded (realtime) response.

136102 The following users are identified for POSIX.1-2024:

136103 • Those employing applications written in high-level languages, such as C, Ada, or
136104 FORTRAN.

136105 • Users who desire conforming applications that do not necessarily require the
136106 characteristics of high-level languages (for example, the speed of execution of compiled
136107 languages or the relative security of source code intellectual property inherent in the
136108 compilation process).

136109 • Users who desire conforming applications that can be developed quickly and can be
136110 modified readily without the use of compilers and other system components that may be
136111 unavailable on small systems or those without special application development
136112 capabilities.

136113 • Users who interact with a system to achieve general-purpose time-sharing capabilities
136114 common to most business or government offices or academic environments: editing, filing,
136115 inter-user communications, printing, and so on.

136116 • Users who develop applications for POSIX-conformant systems.

136117 • Users who develop applications for UNIX systems.

136118 An acknowledged restriction on applicable users is that they are limited to the group of
136119 individuals who are familiar with the style of interaction characteristic of historically-derived
136120 systems based on one of the UNIX operating systems (as opposed to other historical systems
136121 with different models, such as MS/DOS, Macintosh, VMS, MVS, and so on). Typical users

would include program developers, engineers, or general-purpose time-sharing users.

The requirements of users of POSIX.1-2024 can be summarized as a single goal: *application source portability*. The requirements of the user are stated in terms of the requirements of portability of applications. This in turn becomes a requirement for a standardized set of syntax and semantics for operations commonly found on many operating systems.

The following sections list the perceived requirements for application portability.

D.1.1 Configuration Interrogation

An application must be able to determine whether and how certain optional features are provided and to identify the system upon which it is running, so that it may appropriately adapt to its environment.

Applications must have sufficient information to adapt to varying behaviors of the system.

D.1.2 Process Management

An application must be able to manage itself, either as a single process or as multiple processes. Applications must be able to manage other processes when appropriate.

Applications must be able to identify, control, create, and delete processes, and there must be communication of information between processes and to and from the system.

Applications must be able to use multiple flows of control with a process (threads) and synchronize operations between these flows of control.

D.1.3 Access to Data

Applications must be able to operate on the data stored on the system, access it, and transmit it to other applications. Information must have protection from unauthorized or accidental access or modification.

D.1.4 Access to the Environment

Applications must be able to access the external environment to communicate their input and results.

D.1.5 Access to Determinism and Performance Enhancements

Applications must have sufficient control of resource allocation to ensure the timeliness of interactions with external objects.

D.1.6 Operating System-Dependent Profile

The capabilities of the operating system may make certain optional characteristics of the base language in effect no longer optional, and this should be specified.

D.1.7 I/O Interaction

The interaction between the C language I/O subsystem (*stdio*) and the I/O subsystem of POSIX.1-2024 must be specified.

D.1.8 Internationalization Interaction

The effects of the environment of POSIX.1-2024 on the internationalization facilities of the C language must be specified.

D.1.9 C-Language Extensions

Certain functions in the C language must be extended to support the additional capabilities provided by POSIX.1-2024.

D.1.10 Command Language

Users should be able to define procedures that combine simple tools and/or applications into higher-level components that perform to the specific needs of the user. The user should be able to store, recall, use, and modify these procedures. These procedures should employ a powerful command language that is used for recurring tasks in conforming applications (scripts) in the same way that it is used interactively to accomplish one-time tasks. The language and the utilities that it uses must be consistent between systems to reduce errors and retraining.

D.1.11 Interactive Facilities

Use the system to accomplish individual tasks at an interactive terminal. The interface should be consistent, intuitive, and offer usability enhancements to increase the productivity of terminal users, reduce errors, and minimize retraining costs. Online documentation or usage assistance should be available.

D.1.12 Accomplish Multiple Tasks Simultaneously

Access applications and interactive facilities from a single terminal without requiring serial execution: switch between multiple interactive tasks; schedule one-time or periodic background work; display the status of all work in progress or scheduled; influence the priority scheduling of work, when authorized.

D.1.13 Complex Data Manipulation

Manipulate data in files in complex ways: sort, merge, compare, translate, edit, format, pattern match, select subsets (strings, columns, fields, rows, and so on). These facilities should be available to both conforming applications and interactive users.

D.1.14 File Hierarchy Manipulation

Create, delete, move/rename, copy, backup/archive, and display files and directories. These facilities should be available to both conforming applications and interactive users.

D.1.15 Locale Configuration

Customize applications and interactive sessions for the cultural and language conventions of the user. Employ a wide variety of standard character encodings. These facilities should be available to both conforming applications and interactive users.

D.1.16 Inter-User Communication

Send messages or transfer files to other users on the same system or other systems on a network. These facilities should be available to both conforming applications and interactive users.

D.1.17 System Environment

Display information about the status of the system (activities of users and their interactive and background work, file system utilization, system time, configuration, and presence of optional facilities) and the environment of the user (terminal characteristics, and so on). Inform the system operator/administrator of problems. Control access to user files and other resources.

D.1.18 Printing

Output files on a variety of output device classes, accessing devices on local or network-connected systems. Control (or influence) the formatting, priority scheduling, and output distribution of work. These facilities should be available to both conforming applications and interactive users.

D.1.19 Software Development

Develop (create and manage source files, compile/interpret, debug) portable open systems applications and package them for distribution to, and updating of, other systems.

D.2 Portability Capabilities

This section describes the significant portability capabilities of POSIX.1-2024 and indicates how the user requirements listed in Section D.1 (on page 3919) are addressed. The capabilities are listed in the same format as the preceding user requirements; they are summarized below:

- Configuration Interrogation
- Process Management
- Access to Data
- Access to the Environment
- Access to Determinism and Performance Enhancements
- Operating System-Dependent Profile
- I/O Interaction
- Internationalization Interaction
- C-Language Extensions
- Command Language
- Interactive Facilities
- Accomplish Multiple Tasks Simultaneously
- Complex Data Manipulation
- File Hierarchy Manipulation
- Locale Configuration
- Inter-User Communication
- System Environment
- Printing
- Software Development

D.2.1 Configuration Interrogation

The *uname*( ) operation provides basic identification of the system. The *sysconf*( ), *pathconf*( ), and *fpathconf*( ) functions and the *getconf* utility provide means to interrogate the implementation to determine how to adapt to the environment in which it is running. These values can be either static (indicating that all instances of the implementation have the same value) or dynamic (indicating that different instances of the implementation have the different values, or that the value may vary for other reasons, such as reconfiguration).

Unsatisfied Requirements

None directly. However, as new areas are added, there will be a need for additional capability in this area.

D.2.2 Process Management

The *fork*( ), *exec* family, *posix_spawn*( ), and *posix_spawnp*( ) functions provide for the creation of new processes or the insertion of new applications into existing processes. The *_Exit*( ), *_exit*( ), *exit*( ), and *abort*( ) functions allow for the termination of a process by itself. The *wait*( ), *waitid*( ), and *waitpid*( ) functions allow one process to deal with the termination of another.

The *times*( ) function allows for basic measurement of times used by a process. Various functions, including *fstat*( ), *getegid*( ), *geteuid*( ), *getgid*( ), *getgrgid*( ), *getgrnam*( ), *getlogin*( ), *getpid*( ), *getppid*( ), *getpwnam*( ), *getpwuid*( ), *getuid*( ), *lstat*( ), and *stat*( ), provide for access to the identifiers of processes and the identifiers and names of owners of processes (and files).

The various functions operating on environment variables provide for communication of information (primarily user-configurable defaults) from a parent to child processes.

The operations on the current working directory control and interrogate the directory from which relative pathname searches start. The *umask*( ) function controls the default protections applied to files created by the process.

The *alarm*( ), *pause*( ), *sleep*( ), *ualarm*( ), and *usleep*( ) operations allow the process to suspend until a timer has expired or to be notified when a period of time has elapsed. The *time*( ) operation interrogates the current time and date.

The signal mechanism provides for communication of events either from other processes or from the environment to the application, and the means for the application to control the effect of these events. The mechanism provides for external termination of a process and for a process to suspend until an event occurs. The mechanism also provides for a value to be associated with an event.

Job control provides a means to group processes and control them as groups, and to control their access to the function between the user and the system (the ``controlling terminal''). It also provides the means to suspend and resume processes.

The Process Scheduling option provides control of the scheduling and priority of a process.

The Message Passing option provides a means for interprocess communication involving small amounts of data.

The Memory Management facilities provide control of memory resources and for the sharing of memory. This functionality is mandatory on POSIX-conforming systems.

The Threads facilities provide multiple flows of control with a process (threads), synchronization between threads (including mutexes, barriers, and spin locks), association of data with threads, and controlled cancellation of threads.

The XSI interprocess communications functionality provide an alternate set of facilities to manipulate semaphores, message queues, and shared memory. These are provided on XSI-conformant systems to support conforming applications developed to run on UNIX systems.

D.2.3 Access to Data

The *open*( ), *close*( ), *fclose*( ), *fopen*( ), *freopen*( ), *pipe*( ), and *pipe2*( ) functions provide for access to files and data. Such files may be regular files, interprocess data channels (pipes), or devices. Additional types of objects in the file system are permitted and are being contemplated for standardization.

The *access*( ), *chmod*( ), *chown*( ), *dup*( ), *dup2*( ), *dup3*( ), *fchmod*( ), *fcntl*( ), *fstat*( ), *ftruncate*( ), *futimens*( ), *lstat*( ), *readlink*( ), *realpath*( ), *stat*( ), and *utimensat*( ) functions allow for control and

interrogation of file and file-related objects (including symbolic links), and their ownership, protections, and timestamps.

The *fgetc*( ), *fputc*( ), *fread*( ), *fseek*( ), *fsetpos*( ), *fwrite*( ), *getc*( ), *getchar*( ), *lseek*( ), *putchar*( ), *putc*( ), *read*( ), and *write*( ) functions provide for data transfer from the application to files (in all their forms).

The *closedir*( ), *link*( ), *mkdir*( ), *opendir*( ), *readdir*( ), *rename*( ), *rmdir*( ), *rewinddir*( ), and *unlink*( ) functions provide for a complete set of operations on directories. Directories can arbitrarily contain other directories, and a single file can be mentioned in more than one directory.

The *faccessat*( ), *openat*( ), *fchmodat*( ), *fchownat*( ), *fstatat*( ), *linkat*( ) *renameat*( ), *readlinkat*( ), *sylimkat*( ), and *unlinkat*( ) functions allow for race-free and thread-safe file access. The motivation for the introduction of these functions was as follows:

- Interfaces taking a pathname may be limited by the maximum length of a pathname ({PATH_MAX}). The absolute path of files can far exceed this length. The alternative solution of changing the working directory and using relative pathnames is not thread-safe.
- A second motivation is that files accessed outside the current working directory are subject to attacks caused by the race condition created by changing any of the elements of the pathnames used.
- A third motivation is to allow application code which makes use of a virtual current working directory for each individual thread. In the alternative model there is only one current working directory for all threads.

The file-locking mechanisms provide for advisory locking (protection during transactions) of ranges of bytes (in effect, records) in a file.

The *confstr*( ), *fpathconf*( ), *pathconf*( ), and *sysconf*( ) functions provide for enquiry as to the behavior of the system where variability is permitted.

The asynchronous input and output functions *aio_cancel*( ), *aio_error*( ), *aio_fsync*( ), *aio_read*( ), *aio_return*( ), *aio_suspend*( ), *aio_write*( ), and *lio_listio*( ) provide for initiation and control of asynchronous data transfers.

The Synchronized Input and Output option provides for assured commitment of data to media.

D.2.4 Access to the Environment

The operations and types in XBD are provided for access to asynchronous serial devices. The primary intended use for these is the controlling terminal for the application (the interaction point between the user and the system). They are general enough to be used to control any asynchronous serial device. The functions are also general enough to be used with many other device types as a user interface when some emulation is provided.

Less detailed access is provided for other device types, but in many instances an application need not know whether an object in the file system is a device or a regular file to operate correctly.

136320 **Unsatisfied Requirements**

136321 Detailed control of common device classes, specifically magnetic tape, is not provided.

136322 ### D.2.5 Bounded (Realtime) Response

136323 The realtime signal functions *sigqueue*( ), *sigtimedwait*( ), and *sigwaitinfo*( ) provide queued signals
136324 and the prioritization of the handling of signals.

136325 The SCHED_FIFO, SCHED_SPORADIC, and SCHED_RR scheduling policies provide control
136326 over processor allocation.

136327 The semaphore functions *sem_clockwait*( ), *sem_close*( ), *sem_destroy*( ), *sem_getvalue*( ), *sem_init*( ),
136328 *sem_open*( ), *sem_post*( ), *sem_timedwait*( ), *sem_trywait*( ), *sem_unlink*( ), and *sem_wait*( ) provide
136329 high-performance synchronization.

136330 The memory management functions provide memory locking for control of memory allocation,
136331 file mapping for high performance, and shared memory for high-performance interprocess
136332 communication. The Message Passing option provides for interprocess communication without
136333 being dependent on shared memory.

136334 The timers functions *clock_getres*( ), *clock_gettime*( ), *clock_settime*( ), *nanosleep*( ), *timer_create*( ),
136335 *timer_delete*( ), *timer_getoverrun*( ), *timer_gettime*( ), and *timer_settime*( ) provide functionality to
136336 manipulate clocks and timers and include a high resolution function called *nanosleep*( ) with a
136337 finer resolution than the *sleep*( ) function.

136338 The timeout functions — *pthread_mutex_clocklock*( ), *pthread_mutex_timedlock*( ),
136339 *pthread_rwlock_clockrdlock*( ), *pthread_rwlock_clockwrlock*( ), *pthread_rwlock_timedrdlock*( ),
136340 *pthread_rwlock_timedwrlock*( ), *sem_clockwait*( ), and *sem_timedwait*( ) — the Typed Memory
136341 Objects option and the Monotonic Clock facility provide further facilities for applications to use
136342 to obtain predictable bounded response.

136343 ### D.2.6 Operating System-Dependent Profile

136344 POSIX.1-2024 makes no distinction between text and binary files. The values of EXIT_SUCCESS
136345 and EXIT_FAILURE are further defined.

136346 **Unsatisfied Requirements**

136347 None known, but the ISO C standard may contain some additional options that could be
136348 specified.

136349 ### D.2.7 I/O Interaction

136350 POSIX.1-2024 defines how each of the ISO C standard *stdio* functions interact with the POSIX.1
136351 operations, typically specifying the behavior in terms of POSIX.1 operations.

Unsatisfied Requirements

None.

D.2.8 Internationalization Interaction

The POSIX.1-2024 environment operations provide a means to define the environment for *setlocale*( ) and time functions such as *ctime*( ). The *tzset*( ) function is provided to set time conversion information.

The *nl_langinfo*( ) function is provided to query locale-specific cultural settings.

The multiple concurrent locale functions *duplocale*( ), *freelocale*( ), *is***_l( )*, *newlocale*( ), *strcasecmp_l*( ), *strcoll_l*( ), *strfmon_l*( ), *strncasecmp_l*( ), *strxfrm_l*( ), *tolower_l*( ), *toupper_l*( ), *towctrans_l*( ), *towlower*( ), *towupper*( ), *uselocale*( ), *wcscasecmp_l*( ), *wcscoll_l*( ), *wcsncasecmp_l*( ), *wcsxfrm_l*( ), *wctrans_l*( ), and *wctype_l*( ) are provide to support per-thread locale information.

Unsatisfied Requirements

None.

D.2.9 C-Language Extensions

The *setjmp*( ) and *longjmp*( ) functions are not defined to be cognizant of the signal masks defined for POSIX.1. The *sigsetjmp*( ) and *siglongjmp*( ) functions are provided to fill this gap.

Unsatisfied Requirements

None.

D.2.10 Command Language

The shell command language, as described in XCU Chapter 2 (on page 2472), is a common language useful in batch scripts, through an API to high-level languages (for the C-Language Binding option, *system*( ) and *popen*( )) and through an interactive terminal (see the *sh* utility). The shell language has many of the characteristics of a high-level language, but it has been designed to be more suitable for user terminal entry and includes interactive debugging facilities. Through the use of pipelining, many complex commands can be constructed from combinations of data filters and other common components. Shell scripts can be created, stored, recalled, and modified by the user with simple editors.

In addition to the basic shell language, the following utilities offer features that simplify and enhance programmatic access to the utilities and provide features normally found only in high-level languages: *basename*, *bc*, *command*, *dirname*, *echo*, *env*, *expr*, *false*, *printf*, *read*, *sleep*, *tee*, *test*, *time**,[10] *true*, *wait*, *xargs*, and all of the special built-in utilities in XCU Section 2.15 (on page 2526).

10. The utilities listed with an asterisk here and later in this section are present only on systems which support the User Portability Utilities option. There may be further restrictions on the utilities offered with various configuration option combinations; see the individual utility descriptions.

136386 **Unsatisfied Requirements**

136387 None.

136388 ### D.2.11 Interactive Facilities

136389 The utilities offer a common style of command-line interface through conformance to the Utility
136390 Syntax Guidelines (see XBD Section 12.2, on page 215) and the common utility defaults (see XCU
136391 Section 1.4, on page 2462). The *sh* utility offers an interactive command-line history and editing
136392 facility.

136393 The following utilities can be used interactively as well as by scripts; *alias*, *fc*, *mailx*, *unalias*, and
136394 *write*.

136395 The following utilities in the User Portability Utilities option provide for interactive use: *ex*, *more*,
136396 and *vi*; the *man* utility offers online access to system documentation.

136397 **Unsatisfied Requirements**

136398 The command line interface to individual utilities is as intuitive and consistent as historical
136399 practice allows. Work underway based on graphical user interfaces may be more suitable for
136400 novice or occasional users of the system.

136401 ### D.2.12 Accomplish Multiple Tasks Simultaneously

136402 The shell command language offers background processing through the asynchronous list
136403 command form; see XCU Section 2.9 (on page 2499).

136404 The *nohup* utility makes background processing more robust and usable.

136405 The *kill* utility can terminate background jobs.

136406 The following utilities support periodic job scheduling, control, and display: *at*, *batch*, *crontab*,
136407 *nice*, *ps*, and *renice*.

136408 When the User Portability Utilities option is supported, the following utilities allow
136409 manipulation of jobs: *bg*, *fg*, and *jobs*.

136410 **Unsatisfied Requirements**

136411 Terminals with multiple windows may be more suitable for some multi-tasking interactive uses
136412 than the job control approach in POSIX.1-2024. See the comments on graphical user interfaces in
136413 Section D.2.11. The *nice* and *renice* utilities do not necessarily take advantage of complex system
136414 scheduling algorithms that are supported by the realtime options within POSIX.1-2024.

136415 ### D.2.13 Complex Data Manipulation

136416 The following utilities address user requirements in this area: *asa*, *awk*, *bc*, *cmp*, *comm*, *csplit*, *cut*,
136417 *dd*, *diff*, *ed*, *ex**, *expand*, *expr*, *find*, *fold*, *grep*, *head*, *join*, *od*, *paste*, *pr*, *printf*, *sed*, *sort*, *split*, *tabs*, *tail*, *tr*,
136418 *unexpand*, *uniq*, *uudecode*, *uuencode*, and *wc*.

Unsatisfied Requirements

Sophisticated text formatting utilities, such as *troff* or *TeX*, are not included. Standards work in the area of SGML may satisfy this.

D.2.14 File Hierarchy Manipulation

The following utilities address user requirements in this area: *basename*, *cd*, *chgrp*, *chmod*, *chown*, *cksum*, *cp*, *dd*, *df*, *diff*, *dirname*, *du*, *find*, *ls*, *ln*, *mkdir*, *mkfifo*, *mv*, *patch*, *pathchk*, *pax*, *pwd*, *rm*, *rmdir*, *test*, and *touch*.

Unsatisfied Requirements

Some graphical user interfaces offer more intuitive file manager components that allow file manipulation through the use of icons for novice users.

D.2.15 Locale Configuration

The standard utilities are affected by the various *LC_* variables to achieve locale-dependent operation: character classification, collation sequences, regular expressions and shell pattern matching, date and time formats, numeric formatting, and monetary formatting. When the POSIX2_LOCALEDEF option is supported, applications can provide their own locale definition files.

The following utilities address user requirements in this area: *date*, *ed*, *ex**, *find*, *grep*, *locale*, *localedef*, *more**, *sed*, *sh*, *sort*, *tr*, *uniq*, and *vi**.

The *iconv*(), *iconv_close*(), and *iconv_open*() functions are available to allow an application to convert character data between supported character sets.

The *gencat* utility and the *catopen*(), *catclose*(), and *catgets*() functions provide for message catalog manipulation.

Unsatisfied Requirements

Some aspects of multi-byte character and state-encoded character encodings have not yet been addressed. The C-language functions, such as *getopt*(), are generally limited to single-byte characters. The effect of the *LC_MESSAGES* variable on message formats is only suggested at this time.

D.2.16 Inter-User Communication

The following utilities address user requirements in this area: *cksum*, *mailx*, *mesg*, *patch*, *pax*, *talk*, *uudecode*, *uuencode*, *who*, and *write*.

The historical UUCP utilities are included as a separate UUCP Utilities option.

136450 **Unsatisfied Requirements**

136451 None.

136452 D.2.17 System Environment

136453 The following utilities address user requirements in this area: *chgrp*, *chmod*, *chown*, *df*, *du*, *env*,
136454 *getconf*, *id*, *logger*, *logname*, *mesg*, *newgrp*, *ps*, *stty*, *tput*, *tty*, *umask*, *uname*, and *who*.

136455 The *closelog*( ), *openlog*( ), *setlogmask*( ), and *syslog*( ) functions provide system logging facilities on
136456 XSI-conformant systems; these are analogous to the *logger* utility.

136457 **Unsatisfied Requirements**

136458 None.

136459 D.2.18 Printing

136460 The following utilities address user requirements in this area: *pr* and *lp*.

136461 **Unsatisfied Requirements**

136462 There are no features to control the formatting or scheduling of the print jobs.

136463 D.2.19 Software Development

136464 The following utilities address user requirements in this area: *ar*, *asa*, *awk*, *c17*, *ctags*, *getconf*,
136465 *getopts*, *lex*, *localedef*, *make*, *nm*, *od*, *patch*, *pax*, *strings*, *strip*, *time*, and *yacc*.

136466 The *system*( ), *popen*( ), *pclose*( ), *regcomp*( ), *regexec*( ), *regerror*( ), *regfree*( ), *fnmatch*( ), *getopt*( ),
136467 *glob*( ), *globfree*( ), *wordexp*( ), and *wordfree*( ) functions allow C-language programmers to access
136468 some of the interfaces used by the utilities, such as argument processing, regular expressions,
136469 and pattern matching.

136470 The SCCS source-code control system utilities are available on systems supporting the XSI
136471 Development option.

136472 **Unsatisfied Requirements**

136473 There are no language-specific development tools related to languages other than C. There is no
136474 data dictionary or other CASE-like development tools.

136475 D.2.20 Future Growth

136476 It is arguable whether or not all functionality to support applications is potentially within the
136477 scope of POSIX.1-2024. As a simple matter of practicality, it cannot be. Areas such as graphics,
136478 application domain-specific functionality, windowing, and so on, should be in unique standards.
136479 As such, they are properly ``Unsatisfied Requirements'' in terms of providing fully conforming
136480 applications, but ones which are outside the scope of POSIX.1-2024.

136481 However, as the standards evolve, certain functionality once considered ``exotic'' enough to be
136482 part of a separate standard become common enough to be included in a core standard such as
136483 this. Realtime and networking, for example, have both moved from separate standards (with
136484 much difficult cross-referencing) into this standard over time, and although no specific areas

have been identified for inclusion in a future version, such inclusions seem likely.

D.3 Profiling Considerations

This section offers guidance to writers of profiles on how the configurable options, limits, and optional behavior of POSIX.1-2024 should be cited in profiles. Profile writers should consult the general guidance in POSIX.0 when writing POSIX Standardized Profiles.

The information in this section is an inclusive list of features that should be considered by profile writers. Subsetting of POSIX.1-2024 should follow XBD Section 2.1.5.1 (on page 20). A set of profiling options is described in Appendix E (on page 3943).

D.3.1 Configuration Options

There are two set of options suggested by POSIX.1-2024: those for POSIX-conforming systems and those for X/Open System Interface (XSI) conformance. The requirements for XSI conformance are documented in the Base Definitions volume of POSIX.1-2024 and not discussed further here, as they superset the POSIX conformance requirements.

D.3.2 Configuration Options (Shell and Utilities)

There are three broad optional configurations for the Shell and Utilities volume of POSIX.1-2024: basic execution system, development system, and user portability interactive system. The options to support these, and other minor configuration options, are listed in XBD Chapter 2 (on page 15). Profile writers should consult the following list and the comments concerning user requirements addressed by various components in Section D.2 (on page 3923).

POSIX2_UPE

The system supports the User Portability Utilities option.

This option is a requirement for a user portability interactive system. It is required frequently except for those systems, such as embedded realtime or dedicated application systems, that support little or no interactive time-sharing work by users or operators. XSI-conformant systems support this option.

POSIX2_SW_DEV

The system supports the Software Development Utilities option.

This option is required by many systems, even those in which actual software development does not occur. The *make* utility, in particular, is required by many application software packages as they are installed onto the system. If POSIX2_C_DEV is supported, POSIX2_SW_DEV is almost a mandatory requirement because of *ar* and *make*.

POSIX2_C_BIND

The system supports the C-Language Bindings option.

This option is required on some implementations developing complex C applications or on any system installing C applications in source form that require the functions in this option. The *system*() and *popen*() functions, in particular, are widely used by applications; the others are rather more specialized.

POSIX2_C_DEV

The system supports the C-Language Development Utilities option.

This option is required by many systems, even those in which actual C-language software

development does not occur. The *c17* utility, in particular, is required by many application software packages as they are installed onto the system. The *lex* and *yacc* utilities are used less frequently.

POSIX2_FORT_RUN
The system supports the FORTRAN Runtime Utilities option.

This option is required for some FORTRAN applications that need the *asa* utility to convert Hollerith printing statement output. It is unknown how frequently this occurs.

POSIX2_LOCALEDEF
The system supports the creation of locales.

This option is needed if applications require their own customized locale definitions to operate. It is presently unknown whether many applications are dependent on this. However, the option is virtually mandatory for systems in which internationalized applications are developed.

XSI-conformant systems support this option.

POSIX2_CHAR_TERM
The system supports at least one terminal type capable of all operations described in POSIX.1-2024.

On systems with POSIX2_UPE, this option is almost always required. It was developed solely to allow certain specialized vendors and user applications to bypass the requirement for general-purpose asynchronous terminal support. For example, an application and system that was suitable for block-mode terminals would not need this option.

XSI-conformant systems support this option.

D.3.3 Configurable Limits

Very few of the limits need to be increased for profiles. No profile can cite lower values.

{POSIX2_BC_BASE_MAX}
{POSIX2_BC_DIM_MAX}
{POSIX2_BC_SCALE_MAX}
{POSIX2_BC_STRING_MAX}
No increase is anticipated for any of these *bc* values, except for very specialized applications involving huge numbers.

{POSIX2_COLL_WEIGHTS_MAX}
Some natural languages with complex collation requirements require an increase from the default 2 to 4; no higher numbers are anticipated.

{POSIX2_EXPR_NEST_MAX}
No increase is anticipated.

{POSIX2_LINE_MAX}
This number is much larger than most historical applications have been able to use. At some future time, applications may be rewritten to take advantage of even larger values.

{POSIX2_RE_DUP_MAX}
No increase is anticipated.

{POSIX2_VERSION}
This is actually not a limit, but a standard version stamp. Generally, a profile should specify XCU Chapter 2 (on page 2472) by name in the normative references section, not this value.

D.3.4 Configuration Options (System Interfaces)

{NGROUPS_MAX}

A non-zero value indicates that the implementation supports supplementary groups.

This option is needed where there is a large amount of shared use of files, but where a certain amount of protection is needed. Many profiles[11] are known to require this option; it should only be required if needed, but it should never be prohibited.

_POSIX_ADVISORY_INFO

The system provides advisory information for file management.

This option allows the application to specify advisory information that can be used to achieve better or even deterministic response time in file manager or input and output operations.

_POSIX_ASYNCHRONOUS_IO

Support for asynchronous input and output is mandatory in POSIX.1-2024.

_POSIX_BARRIERS

Support for barrier synchronization is mandatory in POSIX.1-2024.

This facility allows efficient synchronization of multiple parallel threads in multi-processor systems in which the operation is supported in part by the hardware architecture.

_POSIX_CHOWN_RESTRICTED

The system restricts the right to ``give away'' files to other users. It is mandatory that an implementation be able to support this facility in POSIX.1-2024; however, it is recognized that implementations need not enable the functionality by default.

Some applications expect that they can change the ownership of files in this way. It is provided where either security or system account requirements cause this ability to be a problem. It is also known to be specified in many profiles.

_POSIX_CLOCK_SELECTION

Support for clock selection is mandatory in POSIX.1-2024.

This facility allows applications to request a high resolution sleep in order to suspend a thread during a relative time interval, or until an absolute time value, using the desired clock. It also allows the application to select the clock used in a *pthread_cond_timedwait*() function call.

_POSIX_CPUTIME

The system supports the Process CPU-Time Clocks option.

This option allows applications to use a new clock that measures the execution times of processes or threads, and the possibility to create timers based upon these clocks, for runtime detection (and treatment) of execution time overruns.

_POSIX_FSYNC

The system supports file synchronization requests.

This option was created to support historical systems that did not provide the feature. Applications that are expecting guaranteed completion of their input and output operations should require the _POSIX_SYNC_IO option. This option should never be prohibited.

XSI-conformant systems support this option.

---

11. There are no formally approved profiles of POSIX.1-2024 at the time of publication; the reference here is to various profiles generated by private bodies or governments.

136611 _POSIX_IPV6
136612 The system supports facilities related to Internet Protocol Version 6 (IPv6).

136613 This option was created to allow systems to transition to IPv6.

136614 _POSIX_JOB_CONTROL
136615 Support for job control is mandatory in POSIX.1-2024.

136616 Most applications that use it can run when it is not present, although with a degraded level
136617 of user convenience.

136618 _POSIX_MAPPED_FILES
136619 Support for memory mapped files is mandatory in POSIX.1-2024.

136620 This facility provides for the mapping of regular files into the process address space.

136621 Both this facility and the Shared Memory Objects option provide shared access to memory
136622 objects in the process address space. The *mmap*() and *munmap*() functions provide the
136623 functionality of existing practice for mapping regular files. This functionality was deemed
136624 unnecessary, if not inappropriate, for embedded systems applications and is expected to be
136625 optional in subprofiles.

136626 _POSIX_MEMLOCK
136627 The system supports the locking of the address space.

136628 This option was created to support historical systems that did not provide the feature. It
136629 should only be required if needed, but it should never be prohibited.

136630 _POSIX_MEMLOCK_RANGE
136631 The system supports the locking of specific ranges of the address space.

136632 For applications that have well-defined sections that need to be locked and others that do
136633 not, POSIX.1-2024 supports an optional set of functions to lock or unlock a range of process
136634 addresses. The following are two reasons for having a means to lock down a specific range:

136635 1. An asynchronous event handler function that must respond to external events in a
136636 deterministic manner such that page faults cannot be tolerated

136637 2. An input/output ``buffer'' area that is the target for direct-to-process I/O, and the
136638 overhead of implicit locking and unlocking for each I/O call cannot be tolerated

136639 It should only be required if needed, but it should never be prohibited.

136640 _POSIX_MEMORY_PROTECTION
136641 Support for memory protection is mandatory in POSIX.1-2024.

136642 The provision of this facility typically imposes additional hardware requirements.

136643 _POSIX_PRIORITIZED_IO
136644 The system provides prioritization for input and output operations.

136645 The use of this option may interfere with the ability of the system to optimize input and
136646 output throughput. It should only be required if needed, but it should never be prohibited.

136647 _POSIX_MESSAGE_PASSING
136648 The system supports the passing of messages between processes.

136649 This option was created to support historical systems that did not provide the feature. The
136650 functionality adds a high-performance XSI interprocess communication facility for local
136651 communication. It should only be required if needed, but it should never be prohibited.

136652 _POSIX_MONOTONIC_CLOCK
136653 Support for a monotonic clock is mandatory in POSIX.1-2024.

136654 This facility allows realtime applications to rely on a monotonically increasing clock that
136655 does not jump backwards, and whose value does not change except for the regular ticking
136656 of the clock.

136657 _POSIX_PRIORITY_SCHEDULING
136658 The system provides priority-based process scheduling.

136659 Support of this option provides predictable scheduling behavior, allowing applications to
136660 determine the order in which processes that are ready to run are granted access to a
136661 processor. It should only be required if needed, but it should never be prohibited.

136662 _POSIX_REALTIME_SIGNALS
136663 Support for realtime signals is mandatory in POSIX.1-2024.

136664 This facility provides prioritized, queued signals with associated data values.

136665 _POSIX_REGEXP
136666 Support for regular expression facilities is mandatory in POSIX.1-2024.

136667 _POSIX_SAVED_IDS
136668 Support for this feature is mandatory in POSIX.1-2024.

136669 Certain classes of applications rely on it for proper operation, and there is no alternative
136670 short of giving the application root privileges on most implementations that did not provide
136671 _POSIX_SAVED_IDS.

136672 _POSIX_SEMAPHORES
136673 Support for counting semaphores is mandatory in POSIX.1-2024.

136674 _POSIX_SHARED_MEMORY_OBJECTS
136675 The system supports the mapping of shared memory objects into the process address space.

136676 Both this option and the Memory Mapped Files option provide shared access to memory
136677 objects in the process address space. The functions defined under this option provide the
136678 functionality of existing practice for shared memory objects. This functionality was deemed
136679 appropriate for embedded systems applications and, hence, is provided under this option.
136680 It should only be required if needed, but it should never be prohibited.

136681 _POSIX_SHELL
136682 Support for the *sh* utility command line interpreter is mandatory in POSIX.1-2024.

136683 _POSIX_SPAWN
136684 The system supports the spawn option.

136685 This option provides applications with an efficient mechanism to spawn execution of a new
136686 process.

136687 _POSIX_SPINLOCKS
136688 Support for spin locks is mandatory in POSIX.1-2024.

136689 This facility provides a simple and efficient synchronization mechanism for threads
136690 executing in multi-processor systems.

136691 _POSIX_SPORADIC_SERVER
136692 The system supports the sporadic server scheduling policy.

136693 This option provides applications with a new scheduling policy for scheduling aperiodic
136694 processes or threads in hard realtime applications.

136695 _POSIX_SYNCHRONIZED_IO
136696 The system supports guaranteed file synchronization.

136697 This option was created to support historical systems that did not provide the feature.
136698 Applications that are expecting guaranteed completion of their input and output operations
136699 should require this option, rather than the File Synchronization option. It should only be
136700 required if needed, but it should never be prohibited.

136701 _POSIX_THREADS
136702 Support for multiple threads of control within a single process is mandatory in
136703 POSIX.1-2024.

136704 _POSIX_THREAD_ATTR_STACKADDR
136705 The system supports specification of the stack address for a created thread.

136706 Applications may take advantage of support of this option for performance benefits, but
136707 dependence on this feature should be minimized. This option should never be prohibited.

136708 XSI-conformant systems support this option.

136709 _POSIX_THREAD_ATTR_STACKSIZE
136710 The system supports specification of the stack size for a created thread.

136711 Applications may require this option in order to ensure proper execution, but such usage
136712 limits portability and dependence on this feature should be minimized. It should only be
136713 required if needed, but it should never be prohibited.

136714 XSI-conformant systems support this option.

136715 _POSIX_THREAD_PRIORITY_SCHEDULING
136716 The system provides priority-based thread scheduling.

136717 Support of this option provides predictable scheduling behavior, allowing applications to
136718 determine the order in which threads that are ready to run are granted access to a processor.
136719 It should only be required if needed, but it should never be prohibited.

136720 _POSIX_THREAD_PRIO_INHERIT
136721 The system provides mutual-exclusion operations with priority inheritance.

136722 Support of this option provides predictable scheduling behavior, allowing applications to
136723 determine the order in which threads that are ready to run are granted access to a processor.
136724 It should only be required if needed, but it should never be prohibited.

136725 _POSIX_THREAD_PRIO_PROTECT
136726 The system supports a priority ceiling emulation protocol for mutual-exclusion operations.

136727 Support of this option provides predictable scheduling behavior, allowing applications to
136728 determine the order in which threads that are ready to run are granted access to a processor.
136729 It should only be required if needed, but it should never be prohibited.

136730 _POSIX_THREAD_PROCESS_SHARED
136731 The system provides shared access among multiple processes to synchronization objects.

136732 This option was created to support historical systems that did not provide the feature. It
136733 should only be required if needed, but it should never be prohibited.

136734 XSI-conformant systems support this option.

136735 _POSIX_THREAD_SAFE_FUNCTIONS
136736 Support for thread-safe functions is mandatory in POSIX.1-2024.

_POSIX_THREAD_SPORADIC_SERVER
The system supports the thread sporadic server scheduling policy.

Support for this option provides applications with a new scheduling policy for scheduling aperiodic threads in hard realtime applications.

_POSIX_TIMEOUTS
Support for timeouts for some blocking services is mandatory in POSIX.1-2024.

_POSIX_TIMERS
Support for higher resolution clocks with multiple timers per process is mandatory in POSIX.1-2024.

This facility is appropriate for applications requiring higher resolution timestamps or needing to control the timing of multiple activities.

_POSIX_TYPED_MEMORY_OBJECTS
The system supports the Typed Memory Objects option.

This option was created to allow realtime applications to access different kinds of physical memory, and allow processes in these applications to share portions of this memory.

D.3.5 Configurable Limits

In general, the configurable limits in the **<limits.h>** header defined in the Base Definitions volume of POSIX.1-2024 have been set to minimal values; many applications or implementations may require larger values. No profile can cite lower values.

{AIO_LISTIO_MAX}
The current minimum is likely to be inadequate for most applications. It is expected that this value will be increased by profiles requiring support for list input and output operations.

{AIO_MAX}
The current minimum is likely to be inadequate for most applications. It is expected that this value will be increased by profiles requiring support for asynchronous input and output operations.

{AIO_PRIO_DELTA_MAX}
The functionality associated with this limit is needed only by sophisticated applications. It is not expected that this limit would need to be increased under a general-purpose profile.

{ARG_MAX}
The current minimum is likely to need to be increased for profiles, particularly as larger amounts of information are passed through the environment. Many implementations are believed to support larger values.

{CHILD_MAX}
The current minimum is suitable only for systems where a single user is not running applications in parallel. It is significantly too low for any system also requiring windows, and if _POSIX_JOB_CONTROL is specified, it should be raised.

{CLOCKRES_MIN}
It is expected that profiles will require a finer granularity clock, perhaps as fine as 1 μs, represented by a value of 1 000 for this limit.

{DELAYTIMER_MAX}
It is believed that most implementations will provide larger values.

136780 {LINK_MAX}
136781 For most applications and usage, the current minimum is adequate. Many implementations
136782 have a much larger value, but this should not be used as a basis for raising the value unless
136783 the applications to be used require it.

136784 {LOGIN_NAME_MAX}
136785 This is not actually a limit, but an implementation parameter. No profile should impose a
136786 requirement on this value.

136787 {MAX_CANON}
136788 For most purposes, the current minimum is adequate. Unless high-speed burst serial
136789 devices are used, it should be left as is.

136790 {MAX_INPUT}
136791 See {MAX_CANON}.

136792 {MQ_OPEN_MAX}
136793 The current minimum should be adequate for most profiles.

136794 {MQ_PRIO_MAX}
136795 The current minimum corresponds to the required number of process scheduling priorities.
136796 Many realtime practitioners believe that the number of message priority levels ought to be
136797 the same as the number of execution scheduling priorities.

136798 {NAME_MAX}
136799 Many implementations now support larger values, and many applications and users
136800 assume that larger names can be used. Many existing profiles also specify a larger value.
136801 Specifying this value will reduce the number of conforming implementations, although this
136802 might not be a significant consideration over time. Values greater than 255 should not be
136803 required.

136804 {NGROUPS_MAX}
136805 The value selected will typically be 8 or larger.

136806 {OPEN_MAX}
136807 The historically common value for this has been 20. Many implementations support larger
136808 values. If applications that use larger values are anticipated, an appropriate value should be
136809 specified.

136810 {PAGESIZE}
136811 This is not actually a limit, but an implementation parameter. No profile should impose a
136812 requirement on this value.

136813 {PATH_MAX}
136814 Historically, the minimum has been either 1 024 or indefinite, depending on the
136815 implementation. Few applications actually require values larger than 256, but some users
136816 may create file hierarchies that must be accessed with longer paths. This value should only
136817 be changed if there is a clear requirement.

136818 {PIPE_BUF}
136819 The current minimum is adequate for most applications. Historically, it has been larger. If
136820 applications that write single transactions larger than this are anticipated, it should be
136821 increased. Applications that write lines of text larger than this probably do not need it
136822 increased, as the text line is delimited by a <newline>.

136823 {POSIX_VERSION}
136824 This is actually not a limit, but a standard version stamp. Generally, a profile should specify
136825 POSIX.1-2024 by a name in the normative references section, not this value.

{PTHREAD_DESTRUCTOR_ITERATIONS}
It is unlikely that applications will need larger values to avoid loss of memory resources.

{PTHREAD_KEYS_MAX}
The current value should be adequate for most profiles.

{PTHREAD_STACK_MIN}
This should not be treated as an actual limit, but as an implementation parameter. No profile should impose a requirement on this value.

{PTHREAD_THREADS_MAX}
It is believed that most implementations will provide larger values.

{RTSIG_MAX}
The current limit was chosen so that the set of POSIX.1 signal numbers can fit within a 32-bit field. It is recognized that most existing implementations define many more signals than are specified in POSIX.1 and, in fact, many implementations have already exceeded 32 signals (including the ``null signal''). Support of {_POSIX_RTSIG_MAX} additional signals may push some implementations over the single 32-bit word line, but is unlikely to push any implementations that are already over that line beyond the 64 signal line.

{SEM_NSEMS_MAX}
The current value should be adequate for most profiles.

{SEM_VALUE_MAX}
The current value should be adequate for most profiles.

{SSIZE_MAX}
This limit reflects fundamental hardware characteristics (the size of an integer), and should not be specified unless it is clearly required. Extreme care should be taken to assure that any value that might be specified does not unnecessarily eliminate implementations because of accidents of hardware design.

{STREAM_MAX}
This limit is very closely related to {OPEN_MAX}. It should never be larger than {OPEN_MAX}, but could reasonably be smaller for application areas where most files are not accessed through *stdio*. Some implementations may limit {STREAM_MAX} to 20 but allow {OPEN_MAX} to be considerably larger. Such implementations should be allowed for if the applications permit.

{TIMER_MAX}
The current limit should be adequate for most profiles, but it may need to be larger for applications with a large number of asynchronous operations.

{TTY_NAME_MAX}
This is not actually a limit, but an implementation parameter. No profile should impose a requirement on this value.

{TZNAME_MAX}
The minimum has been historically adequate, but if longer timezone names are anticipated (particularly such values as UTC−1), this should be increased.

D.3.6 Optional Behavior

In POSIX.1-2024, there are no instances of the terms unspecified, undefined, implementation-defined, or with the verbs ``may'' or ``need not'', that the standard developers anticipate or sanction as suitable for profile or test method citation. All of these are merely warnings to conforming applications to avoid certain areas that can vary from system to system, and even over time on the same system. In many cases, these terms are used explicitly to support extensions, but profiles should not anticipate and require such extensions; future versions of this standard may do so.

Rationale (Informative)

Part E:

Subprofiling Considerations

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

136879 *Appendix E*

136880 # *Subprofiling Considerations (Informative)*

136881 This section contains further information to satisfy the requirement that the project scope enable
136882 subprofiling of POSIX.1-2024. The approach taken is to include a general requirement in
136883 normative text regarding subprofiling and to include an informative section (here) containing a
136884 proposed set of subprofiling options.

136885 ## E.1 Subprofiling Option Groups

136886 The following Option Groups[12] are defined to support profiling. Systems claiming support to
136887 POSIX.1-2024 need not implement these options apart from the requirements stated in XBD
136888 Section 2.1.3 (on page 17). These Option Groups allow profiles to subset the System Interfaces
136889 volume of POSIX.1-2024 by collecting sets of related functions and generic functions.

136890 POSIX_ASYNCHRONOUS_IO: Asynchronous Input and Output Functions
136891 *aio_cancel*( ), *aio_error*( ), *aio_fsync*( ), *aio_read*( ), *aio_return*( ), *aio_suspend*( ), *aio_write*( ),
136892 *lio_listio*( )

136893 POSIX_BARRIERS: Barriers
136894 *pthread_barrier_destroy*( ), *pthread_barrier_init*( ), *pthread_barrier_wait*( ),
136895 *pthread_barrierattr_destroy*( ), *pthread_barrierattr_getpshared*( ), *pthread_barrierattr_init*( ),
136896 *pthread_barrierattr_setpshared*( )

136897 POSIX_C_LANG_ATOMICS: ISO C Atomic Operations
136898 *atomic_compare_exchange_strong*( ), *atomic_compare_exchange_strong_explicit*( ),
136899 *atomic_compare_exchange_weak*( ), *atomic_compare_exchange_weak_explicit*( ), *atomic_exchange*( ),
136900 *atomic_exchange_explicit*( ), *atomic_fetch_add*( ), *atomic_fetch_add_explicit*( ), *atomic_fetch_and*( ),
136901 *atomic_fetch_and_explicit*( ), *atomic_fetch_or*( ), *atomic_fetch_or_explicit*( ), *atomic_fetch_sub*( ),
136902 *atomic_fetch_sub_explicit*( ), *atomic_fetch_xor*( ), *atomic_fetch_xor_explicit*( ), *atomic_flag_clear*( ),
136903 *atomic_flag_clear_explicit*( ), *atomic_flag_test_and_set*( ), *atomic_flag_test_and_set_explicit*( ),
136904 *atomic_init*( ), *atomic_is_lock_free*( ), *atomic_load*( ), *atomic_load_explicit*( ), *atomic_signal_fence*( ),
136905 *atomic_thread_fence*( ), *atomic_store*( ), *atomic_store_explicit*( ), *kill_dependency*( )

136906 POSIX_C_LANG_JUMP: Jump Functions
136907 *longjmp*( ), *setjmp*( )

136908 POSIX_C_LANG_MATH: Maths Library
136909 *CMPLX*( ), *CMPLXF*( ), *CMPLXL*( ), *acos*( ), *acosf*( ), *acosh*( ), *acoshf*( ), *acoshl*( ), *acosl*( ), *asin*( ),
136910 *asinf*( ), *asinh*( ), *asinhf*( ), *asinhl*( ), *asinl*( ), *atan*( ), *atan2*( ), *atan2f*( ), *atan2l*( ), *atanf*( ), *atanh*( ),
136911 *atanhf*( ), *atanhl*( ), *atanl*( ), *cabs*( ), *cabsf*( ), *cabsl*( ), *cacos*( ), *cacosf*( ), *cacosh*( ), *cacoshf*( ),
136912 *cacoshl*( ), *cacosl*( ), *carg*( ), *cargf*( ), *cargl*( ), *casin*( ), *casinf*( ), *casinh*( ), *casinhf*( ), *casinhl*( ),
136913 *casinl*( ), *catan*( ), *catanf*( ), *catanh*( ), *catanhf*( ), *catanhl*( ), *catanl*( ), *cbrt*( ), *cbrtf*( ), *cbrtl*( ), *ccos*( ),
136914 *ccosf*( ), *ccosh*( ), *ccoshf*( ), *ccoshl*( ), *ccosl*( ), *ceil*( ), *ceilf*( ), *ceill*( ), *cexp*( ), *cexpf*( ), *cexpl*( ), *cimag*( ),
136915 *cimagf*( ), *cimagl*( ), *clog*( ), *clogf*( ), *clogl*( ), *conj*( ), *conjf*( ), *conjl*( ), *copysign*( ), *copysignf*( ),
136916 *copysignl*( ), *cos*( ), *cosf*( ), *cosh*( ), *coshf*( ), *coshl*( ), *cosl*( ), *cpow*( ), *cpowf*( ), *cpowl*( ), *cproj*( ),
136917 *cprojf*( ), *cprojl*( ), *creal*( ), *crealf*( ), *creall*( ), *csin*( ), *csinf*( ), *csinh*( ), *csinhf*( ), *csinhl*( ), *csinl*( ),
136918 *csqrt*( ), *csqrtf*( ), *csqrtl*( ), *ctan*( ), *ctanf*( ), *ctanh*( ), *ctanhf*( ), *ctanhl*( ), *ctanl*( ), *erf*( ), *erfc*( ), *erfcf*( ),

136919 12. These are modeled on the Units of Functionality from IEEE Std 1003.13-1998.

erfcl( ), *erff*( ), *erfl*( ), *exp*( ), *exp2*( ), *exp2f*( ), *exp2l*( ), *expf*( ), *expl*( ), *expm1*( ), *expm1f*( ), *expm1l*( ), *fabs*( ), *fabsf*( ), *fabsl*( ), *fdim*( ), *fdimf*( ), *fdiml*( ), *floor*( ), *floorf*( ), *floorl*( ), *fma*( ), *fmaf*( ), *fmal*( ), *fmax*( ), *fmaxf*( ), *fmaxl*( ), *fmin*( ), *fminf*( ), *fminl*( ), *fmod*( ), *fmodf*( ), *fmodl*( ), *fpclassify*( ), *frexp*( ), *frexpf*( ), *frexpl*( ), *hypot*( ), *hypotf*( ), *hypotl*( ), *ilogb*( ), *ilogbf*( ), *ilogbl*( ), *isfinite*( ), *isgreater*( ), *isgreaterequal*( ), *isinf*( ), *isless*( ), *islessequal*( ), *islessgreater*( ), *isnan*( ), *isnormal*( ), *isunordered*( ), *ldexp*( ), *ldexpf*( ), *ldexpl*( ), *lgamma*( ), *lgammaf*( ), *lgammal*( ), *llrint*( ), *llrintf*( ), *llrintl*( ), *llround*( ), *llroundf*( ), *llroundl*( ), *log*( ), *log10*( ), *log10f*( ), *log10l*( ), *log1p*( ), *log1pf*( ), *log1pl*( ), *log2*( ), *log2f*( ), *log2l*( ), *logb*( ), *logbf*( ), *logbl*( ), *logf*( ), *logl*( ), *lrint*( ), *lrintf*( ), *lrintl*( ), *lround*( ), *lroundf*( ), *lroundl*( ), *modf*( ), *modff*( ), *modfl*( ), *nan*( ), *nanf*( ), *nanl*( ), *nearbyint*( ), *nearbyintf*( ), *nearbyintl*( ), *nextafter*( ), *nextafterf*( ), *nextafterl*( ), *nexttoward*( ), *nexttowardf*( ), *nexttowardl*( ), *pow*( ), *powf*( ), *powl*( ), *remainder*( ), *remainderf*( ), *remainderl*( ), *remquo*( ), *remquof*( ), *remquol*( ), *rint*( ), *rintf*( ), *rintl*( ), *round*( ), *roundf*( ), *roundl*( ), *scalbln*( ), *scalblnf*( ), *scalblnl*( ), *scalbn*( ), *scalbnf*( ), *scalbnl*( ), *signbit*( ), *sin*( ), *sinf*( ), *sinh*( ), *sinhf*( ), *sinhl*( ), *sinl*( ), *sqrt*( ), *sqrtf*( ), *sqrtl*( ), *tan*( ), *tanf*( ), *tanh*( ), *tanhf*( ), *tanhl*( ), *tanl*( ), *tgamma*( ), *tgammaf*( ), *tgammal*( ), *trunc*( ), *truncf*( ), *truncl*( )

POSIX_C_LANG_SUPPORT: General ISO C Library

abs( ), *aligned_alloc*( ), *asctime*( ), *atof*( ), *atoi*( ), *atol*( ), *atoll*( ), *bsearch*( ), *calloc*( ), *ctime*( ), *difftime*( ), *div*( ), *feclearexcept*( ), *fegetenv*( ), *fegetexceptflag*( ), *fegetround*( ), *feholdexcept*( ), *feraiseexcept*( ), *fesetenv*( ), *fesetexceptflag*( ), *fesetround*( ), *fetestexcept*( ), *feupdateenv*( ), *free*( ), *gmtime*( ), *imaxabs*( ), *imaxdiv*( ), *isalnum*( ), *isalpha*( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *labs*( ), *ldiv*( ), *llabs*( ), *lldiv*( ), *localeconv*( ), *localtime*( ), *malloc*( ), *memchr*( ), *memcmp*( ), *memcpy*( ), *memmove*( ), *memset*( ), *mktime*( ), *qsort*( ), *rand*( ), *realloc*( ), *setlocale*( ), *snprintf*( ), *sprintf*( ), *srand*( ), *sscanf*( ), *strcat*( ), *strchr*( ), *strcmp*( ), *strcoll*( ), *strcpy*( ), *strcspn*( ), *strerror*( ), *strftime*( ), *strlen*( ), *strncat*( ), *strncmp*( ), *strncpy*( ), *strpbrk*( ), *strrchr*( ), *strspn*( ), *strstr*( ), *strtod*( ), *strtof*( ), *strtoimax*( ), *strtok*( ), *strtol*( ), *strtold*( ), *strtoll*( ), *strtoul*( ), *strtoull*( ), *strtoumax*( ), *strxfrm*( ), *time*( ), *timespec_get*( ), *tolower*( ), *toupper*( ), *tzname*, *tzset*( ), *va_arg*( ), *va_copy*( ), *va_end*( ), *va_start*( ), *vsnprintf*( ), *vsprintf*( ), *vsscanf*( )

POSIX_C_LANG_SUPPORT_R: Thread-Safe General ISO C Library

gmtime_r( ), *localtime_r*( ), *qsort_r*( ), *strerror_r*( ), *strtok_r*( )

POSIX_C_LANG_THREADS: ISO C Threads

call_once( ), *cnd_broadcast*( ), *cnd_signal*( ), *cnd_destroy*( ), *cnd_init*( ), *cnd_timedwait*( ), *cnd_wait*( ), *mtx_destroy*( ), *mtx_init*( ), *mtx_lock*( ), *mtx_timedlock*( ), *mtx_trylock*( ), *mtx_unlock*( ), *thrd_create*( ), *thrd_current*( ), *thrd_detach*( ), *thrd_equal*( ), *thrd_exit*( ), *thrd_join*( ), *thrd_sleep*( ), *thrd_yield*( ), *tss_create*( ), *tss_delete*( ), *tss_get*( ), *tss_set*( )

POSIX_C_LANG_UCHAR: ISO C Unicode Utilities

c16rtomb( ), *c32rtomb*( ), *mbrtoc16*( ), *mbrtoc32*( )

POSIX_C_LANG_WIDE_CHAR: Wide-Character ISO C Library

btowc( ), *iswalnum*( ), *iswalpha*( ), *iswblank*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *mblen*( ), *mbrlen*( ), *mbrtowc*( ), *mbsinit*( ), *mbsrtowcs*( ), *mbstowcs*( ), *mbtowc*( ), *swprintf*( ), *swscanf*( ), *towctrans*( ), *towlower*( ), *towupper*( ), *vswprintf*( ), *vswscanf*( ), *wcrtomb*( ), *wcscat*( ), *wcschr*( ), *wcscmp*( ), *wcscoll*( ), *wcscpy*( ), *wcscspn*( ), *wcsftime*( ), *wcslen*( ), *wcsncat*( ), *wcsncmp*( ), *wcsncpy*( ), *wcspbrk*( ), *wcsrchr*( ), *wcsrtombs*( ), *wcsspn*( ), *wcsstr*( ), *wcstod*( ), *wcstof*( ), *wcstoimax*( ), *wcstok*( ), *wcstol*( ), *wcstold*( ), *wcstoll*( ), *wcstombs*( ), *wcstoul*( ), *wcstoull*( ), *wcstoumax*( ), *wcsxfrm*( ), *wctob*( ), *wctomb*( ), *wctrans*( ), *wctype*( ), *wmemchr*( ), *wmemcmp*( ), *wmemcpy*( ), *wmemmove*( ), *wmemset*( )

POSIX_C_LANG_WIDE_CHAR_EXT: Extended Wide-Character ISO C Library

mbsnrtowcs( ), *wcpcpy*( ), *wcpncpy*( ), *wcscasecmp*( ), *wcsdup*( ), *wcslcat*( ), *wcslcpy*( ), *wcsncasecmp*( ), *wcsnlen*( ), *wcsnrtombs*( )

136970 POSIX_C_LIB_EXT: General C Library Extension
136971 *fnmatch*( ), *getentropy*( ), *getopt*( ), *getsubopt*( ), *memmem*( ), *optarg*, *opterr*, *optind*, *optopt*,
136972 *reallocarray*( ), *stpcpy*( ), *stpncpy*( ), *strcasecmp*( ), *strdup*( ), *strfmon*( ), *strlcat*( ), *strlcpy*( ),
136973 *strncasecmp*( ), *strndup*( ), *strnlen*( )

136974 POSIX_CLOCK_SELECTION: Clock Selection
136975 *clock_nanosleep*( ), *pthread_condattr_getclock*( ), *pthread_condattr_setclock*( )

136976 POSIX_DEVICE_IO: Device Input and Output
136977 *FD_CLR*( ), *FD_ISSET*( ), *FD_SET*( ), *FD_ZERO*( ), *clearerr*( ), *close*( ), *fclose*( ), *fdopen*( ), *feof*( ),
136978 *ferror*( ), *fflush*( ), *fgetc*( ), *fgets*( ), *fileno*( ), *fopen*( ), *fprintf*( ), *fputc*( ), *fputs*( ), *fread*( ), *freopen*( ),
136979 *fscanf*( ), *fwrite*( ), *getc*( ), *getchar*( ), *open*( ), *perror*( ), *poll*( ), *posix_close*( ), *ppoll*( ), *printf*( ),
136980 *pread*( ), *pselect*( ), *putc*( ), *putchar*( ), *puts*( ), *pwrite*( ), *read*( ), *scanf*( ), *select*( ), *setbuf*( ),
136981 *setvbuf*( ), *stderr*, *stdin*, *stdout*, *ungetc*( ), *vfprintf*( ), *vfscanf*( ), *vprintf*( ), *vscanf*( ), *write*( )

136982 POSIX_DEVICE_IO_EXT: Extended Device Input and Output
136983 *asprintf*( ), *dprintf*( ), *fmemopen*( ), *open_memstream*( ), *vasprintf*( ), *vdprintf*( )

136984 POSIX_DEVICE_SPECIFIC: General Terminal
136985 *cfgetispeed*( ), *cfgetospeed*( ), *cfsetispeed*( ), *cfsetospeed*( ), *ctermid*( ), *isatty*( ), *tcdrain*( ), *tcflow*( ),
136986 *tcflush*( ), *tcgetattr*( ), *tcgetwinsize*( ), *tcsendbreak*( ), *tcsetattr*( ), *tcsetwinsize*( ), *ttyname*( )

136987 POSIX_DEVICE_SPECIFIC_R: Thread-Safe General Terminal
136988 *ttyname_r*( )

136989 POSIX_DYNAMIC_LINKING: Dynamic Linking
136990 *dladdr*( ), *dlclose*( ), *dlerror*( ), *dlopen*( ), *dlsym*( )

136991 POSIX_FD_MGMT: File Descriptor Management
136992 *dup*( ), *dup2*( ), *dup3*( ), *fcntl*( ), *fgetpos*( ), *fseek*( ), *fseeko*( ), *fsetpos*( ), *ftell*( ), *ftello*( ), *ftruncate*( ),
136993 *lseek*( ), *rewind*( )

136994 POSIX_FIFO: FIFO
136995 *mkfifo*( )

136996 POSIX_FIFO_FD: FIFO File Descriptor Routines
136997 *mkfifoat*( ), *mknodat*( )

136998 POSIX_FILE_ATTRIBUTES: File Attributes
136999 *chmod*( ), *chown*( ), *fchmod*( ), *fchown*( ), *umask*( )

137000 POSIX_FILE_ATTRIBUTES_FD: File Attributes File Descriptor Routines
137001 *fchmodat*( ), *fchownat*( )

137002 POSIX_FILE_LOCKING: Thread-Safe Stdio Locking
137003 *flockfile*( ), *ftrylockfile*( ), *funlockfile*( ), *getc_unlocked*( ), *getchar_unlocked*( ), *putc_unlocked*( ),
137004 *putchar_unlocked*( )

137005 POSIX_FILE_SYSTEM: File System
137006 *access*( ), *chdir*( ), *closedir*( ), *creat*( ), *fchdir*( ), *fpathconf*( ), *fstat*( ), *fstatvfs*( ), *futimens*( ), *getcwd*( ),
137007 *link*( ), *mkdir*( ), *mkostemp*( ), *mkstemp*( ), *opendir*( ), *pathconf*( ), *posix_getdents*( ), *readdir*( ),
137008 *remove*( ), *rename*( ), *rewinddir*( ), *rmdir*( ), *stat*( ), *statvfs*( ), *tmpfile*( ), *tmpnam*( ), *truncate*( ),
137009 *unlink*( )

137010 POSIX_FILE_SYSTEM_EXT: File System Extensions
137011 *alphasort*( ), *dirfd*( ), *getdelim*( ), *getline*( ), *mkdtemp*( ), *scandir*( )

137012 POSIX_FILE_SYSTEM_FD: File System File Descriptor Routines
137013 *faccessat*( ), *fdopendir*( ), *fstatat*( ), *linkat*( ), *mkdirat*( ), *openat*( ), *renameat*( ), *unlinkat*( ),
137014 *utimensat*( )

137015 POSIX_FILE_SYSTEM_GLOB: File System Glob Expansion
137016 *glob*( ), *globfree*( )

137017 POSIX_FILE_SYSTEM_R: Thread-Safe File System
137018 *readdir_r*( )

137019 POSIX_I18N: Internationalization
137020 *bind_textdomain_codeset*( ), *bindtextdomain*( ), *catclose*( ), *catgets*( ), *catopen*( ), *dcgettext*( ),
137021 *dcgettext_l*( ), *dcngettext*( ), *dcngettext_l*( ), *dgettext*( ), *dgettext_l*( ), *dngettext*( ), *dngettext_l*( ),
137022 *gettext*( ), *gettext_l*( ), *iconv*( ), *iconv_close*( ), *iconv_open*( ), *ngettext*( ), *ngettext_l*( ),
137023 *nl_langinfo*( ), *nl_langinfo_l*( ), *textdomain*( )

137024 POSIX_JOB_CONTROL: Job Control
137025 *setpgid*( ), *tcgetpgrp*( ), *tcsetpgrp*( ), *tcgetsid*( )

137026 POSIX_MAPPED_FILES: Memory Mapped Files
137027 *mmap*( ), *munmap*( )

137028 POSIX_MEMORY_PROTECTION: Memory Protection
137029 *mprotect*( )

137030 POSIX_MULTI_CONCURRENT_LOCALES: Multiple Concurrent Locales
137031 *duplocale*( ), *freelocale*( ), *getlocalename_l*( ), *isalnum_l*( ), *isalpha_l*( ), *isblank_l*( ), *iscntrl_l*( ),
137032 *isdigit_l*( ), *isgraph_l*( ), *islower_l*( ), *isprint_l*( ), *ispunct_l*( ), *isspace_l*( ), *isupper_l*( ),
137033 *iswalnum_l*( ), *iswalpha_l*( ), *iswblank_l*( ), *iswcntrl_l*( ), *iswctype_l*( ), *iswdigit_l*( ), *iswgraph_l*( ),
137034 *iswlower_l*( ), *iswprint_l*( ), *iswpunct_l*( ), *iswspace_l*( ), *iswupper_l*( ), *iswxdigit_l*( ), *isxdigit_l*( ),
137035 *newlocale*( ), *strcasecmp_l*( ), *strcoll_l*( ), *strerror_l*( ), *strfmon_l*( ), *strftime_l*( ), *strncasecmp_l*( ),
137036 *strxfrm_l*( ), *tolower_l*( ), *toupper_l*( ), *towctrans_l*( ), *towlower_l*( ), *towupper_l*( ), *uselocale*( ),
137037 *wcscasecmp_l*( ), *wcscoll_l*( ), *wcsncasecmp_l*( ), *wcsxfrm_l*( ), *wctrans_l*( ), *wctype_l*( )

137038 POSIX_MULTI_PROCESS: Multiple Processes
137039 *_Exit*( ), *_Fork*( ), *_exit*( ), *assert*( ), *at_quick_exit*( ), *atexit*( ), *clock*( ), *execl*( ), *execle*( ), *execlp*( ),
137040 *execv*( ), *execve*( ), *execvp*( ), *exit*( ), *fork*( ), *getpgrp*( ), *getpgid*( ), *getpid*( ), *getppid*( ), *getrlimit*( ),
137041 *getsid*( ), *quick_exit*( ), *setrlimit*( ), *setsid*( ), *sleep*( ), *times*( ), *wait*( ), *waitid*( ), *waitpid*( )

137042 POSIX_MULTI_PROCESS_FD: Multiple Processes File Descriptor Routines
137043 *fexecve*( )

137044 POSIX_NETWORKING: Networking
137045 *accept*( ), *accept4*( ), *be16toh*( ), *be32toh*( ), *be64toh*( ), *bind*( ), *connect*( ), *endhostent*( ), *endnetent*( ),
137046 *endprotoent*( ), *endservent*( ), *freeaddrinfo*( ), *gai_strerror*( ), *getaddrinfo*( ), *gethostent*( ),
137047 *gethostname*( ), *getnameinfo*( ), *getnetbyaddr*( ), *getnetbyname*( ), *getnetent*( ), *getpeername*( ),
137048 *getprotobyname*( ), *getprotobynumber*( ), *getprotoent*( ), *getservbyname*( ), *getservbyport*( ),
137049 *getservent*( ), *getsockname*( ), *getsockopt*( ), *htobe16*( ), *htobe32*( ), *htobe64*( ), *htole16*( ), *htole32*( ),
137050 *htole64*( ), *htonl*( ), *htons*( ), *if_freenameindex*( ), *if_indextoname*( ), *if_nameindex*( ),
137051 *if_nametoindex*( ), *inet_addr*( ), *inet_ntoa*( ), *inet_ntop*( ), *inet_pton*( ), *le16toh*( ), *le32toh*( ),
137052 *le64toh*( ), *listen*( ), *ntohl*( ), *ntohs*( ), *recv*( ), *recvfrom*( ), *recvmsg*( ), *send*( ), *sendmsg*( ), *sendto*( ),
137053 *sethostent*( ), *setnetent*( ), *setprotoent*( ), *setservent*( ), *setsockopt*( ), *shutdown*( ), *socket*( ),
137054 *sockatmark*( ), *socketpair*( )

137055 POSIX_PIPE: Pipe
137056 *pipe*( ), *pipe2*( )

137057 POSIX_ROBUST_MUTEXES: Robust Mutexes
137058 *pthread_mutex_consistent*( ), *pthread_mutexattr_getrobust*( ), *pthread_mutexattr_setrobust*( )

137059 POSIX_REALTIME_SIGNALS: Realtime Signals
137060 *sigqueue*( ), *sigtimedwait*( ), *sigwaitinfo*( )

137061 POSIX_REGEXP: Regular Expressions
137062 *regcomp*( ), *regerror*( ), *regexec*( ), *regfree*( )

137063 POSIX_RW_LOCKS: Reader Writer Locks
137064 *pthread_rwlock_clockrdlock*( ), *pthread_rwlock_clockwrlock*( ), *pthread_rwlock_destroy*( ),
137065 *pthread_rwlock_init*( ), *pthread_rwlock_rdlock*( ), *pthread_rwlock_timedrdlock*( ),
137066 *pthread_rwlock_timedwrlock*( ), *pthread_rwlock_tryrdlock*( ), *pthread_rwlock_trywrlock*( ),
137067 *pthread_rwlock_unlock*( ), *pthread_rwlock_wrlock*( ), *pthread_rwlockattr_destroy*( ),
137068 *pthread_rwlockattr_init*( ), *pthread_rwlockattr_getpshared*( ), *pthread_rwlockattr_setpshared*( )

137069 POSIX_SEMAPHORES: Semaphores
137070 *sem_clockwait*( ), *sem_close*( ), *sem_destroy*( ), *sem_getvalue*( ), *sem_init*( ), *sem_open*( ),
137071 *sem_post*( ), *sem_timedwait*( ), *sem_trywait*( ), *sem_unlink*( ), *sem_wait*( )

137072 POSIX_SHELL_FUNC: Shell and Utilities
137073 *pclose*( ), *popen*( ), *system*( ), *wordexp*( ), *wordfree*( )

137074 POSIX_SIGNAL_JUMP: Signal Jump Functions
137075 *siglongjmp*( ), *sigsetjmp*( )

137076 POSIX_SIGNALS: Signals
137077 *abort*( ), *alarm*( ), *kill*( ), *pause*( ), *raise*( ), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigemptyset*( ),
137078 *sigfillset*( ), *sigismember*( ), *signal*( ), *sigpending*( ), *sigprocmask*( ), *sigsuspend*( ), *sigwait*( )

137079 POSIX_SIGNALS_EXT: Extended Signals
137080 *psignal*( ), *psiginfo*( ), *sig2str*( ), *str2sig*( ), *strsignal*( )

137081 POSIX_SINGLE_PROCESS: Single Process
137082 *confstr*( ), *environ*, *errno*, *getenv*( ), *secure_getenv*( ), *setenv*( ), *sysconf*( ), *uname*( ), *unsetenv*( )

137083 POSIX_SPIN_LOCKS: Spin Locks
137084 *pthread_spin_destroy*( ), *pthread_spin_init*( ), *pthread_spin_lock*( ), *pthread_spin_trylock*( ),
137085 *pthread_spin_unlock*( )

137086 POSIX_SYMBOLIC_LINKS: Symbolic Links
137087 *lchown*( ),[13] *lstat*( ), *readlink*( ), *realpath*( ), *symlink*( )

137088 POSIX_SYMBOLIC_LINKS_FD: Symbolic Links File Descriptor Routines
137089 *readlinkat*( ), *symlinkat*( )

137090 POSIX_SYSTEM_DATABASE: System Database
137091 *getgrgid*( ), *getgrnam*( ), *getpwnam*( ), *getpwuid*( )

137092 POSIX_SYSTEM_DATABASE_R: Thread-Safe System Database
137093 *getgrgid_r*( ), *getgrnam_r*( ), *getpwnam_r*( ), *getpwuid_r*( )

137094 POSIX_THREADS_BASE: Base Threads
137095 *pthread_atfork*( ), *pthread_attr_destroy*( ), *pthread_attr_getdetachstate*( ),
137096 *pthread_attr_getschedparam*( ), *pthread_attr_init*( ), *pthread_attr_setdetachstate*( ),
137097 *pthread_attr_setschedparam*( ), *pthread_cancel*( ), *pthread_cleanup_pop*( ), *pthread_cleanup_push*( ),
137098 *pthread_cond_broadcast*( ), *pthread_cond_clockwait*( ), *pthread_cond_destroy*( ),
137099 *pthread_cond_init*( ), *pthread_cond_signal*( ), *pthread_cond_timedwait*( ), *pthread_cond_wait*( ),
137100 *pthread_condattr_destroy*( ), *pthread_condattr_init*( ), *pthread_create*( ), *pthread_detach*( ),
137101 *pthread_equal*( ), *pthread_exit*( ), *pthread_getspecific*( ), *pthread_join*( ), *pthread_key_create*( ),
137102 *pthread_key_delete*( ), *pthread_kill*( ), *pthread_mutex_clocklock*( ), *pthread_mutex_destroy*( ),
137103 *pthread_mutex_init*( ), *pthread_mutex_lock*( ), *pthread_mutex_timedlock*( ),
137104 *pthread_mutex_trylock*( ), *pthread_mutex_unlock*( ), *pthread_mutexattr_destroy*( ),

---

137105 13. The *lchown*( ) function also depends on POSIX_FILE_ATTRIBUTES.

137106 *pthread_mutexattr_init*( ), *pthread_once*( ), *pthread_self*( ), *pthread_setcancelstate*( ),
137107 *pthread_setcanceltype*( ), *pthread_setspecific*( ), *pthread_sigmask*( ), *pthread_testcancel*( ),
137108 *sched_yield*( )

137109 POSIX_THREADS_EXT: Extended Threads
137110 *pthread_attr_getguardsize*( ), *pthread_attr_setguardsize*( ), *pthread_mutexattr_gettype*( ),
137111 *pthread_mutexattr_settype*( )

137112 POSIX_TIMERS: Timers
137113 *clock_getres*( ), *clock_gettime*( ), *clock_settime*( ), *nanosleep*( ), *timer_create*( ), *timer_delete*( ),
137114 *timer_getoverrun*( ), *timer_gettime*( ), *timer_settime*( )

137115 POSIX_USER_GROUPS: User and Group
137116 *getegid*( ), *geteuid*( ), *getgid*( ), *getgroups*( ), *getlogin*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ),
137117 *setuid*( )

137118 POSIX_USER_GROUPS_R: Thread-Safe User and Group
137119 *getlogin_r*( )

137120 POSIX_WIDE_CHAR_DEVICE_IO: Device Input and Output
137121 *fgetwc*( ), *fgetws*( ), *fputwc*( ), *fputws*( ), *fwide*( ), *fwprintf*( ), *fwscanf*( ), *getwc*( ), *getwchar*( ),
137122 *open_wmemstream*( ), *putwc*( ), *putwchar*( ), *ungetwc*( ), *vfwprintf*( ), *vfwscanf*( ), *vwprintf*( ),
137123 *vwscanf*( ), *wprintf*( ), *wscanf*( )

137124 XSI_C_LANG_SUPPORT: XSI General C Library
137125 *a64l*( ), *daylight*, *drand48*( ), *erand48*( ), *ffs*( ), *ffsl*( ), *ffsll*( ), *getdate*( ), *hcreate*( ), *hdestroy*( ),
137126 *hsearch*( ), *initstate*( ), *insque*( ), *jrand48*( ), *l64a*( ), *lcong48*( ), *lfind*( ), *lrand48*( ), *lsearch*( ),
137127 *memccpy*( ), *mrand48*( ), *nrand48*( ), *random*( ), *remque*( ), *seed48*( ), *setstate*( ), *signgam*,
137128 *srand48*( ), *srandom*( ), *strptime*( ), *swab*( ), *tdelete*( ), *tfind*( ), *timezone*, *tsearch*( ), *twalk*( )

137129 XSI_DBM: XSI Database Management
137130 *dbm_clearerr*( ), *dbm_close*( ), *dbm_delete*( ), *dbm_error*( ), *dbm_fetch*( ), *dbm_firstkey*( ),
137131 *dbm_nextkey*( ), *dbm_open*( ), *dbm_store*( )

137132 XSI_DEVICE_IO: XSI Device Input and Output
137133 *fmtmsg*( ), *readv*( ), *writev*( )

137134 XSI_DEVICE_SPECIFIC: XSI General Terminal
137135 *grantpt*( ), *posix_openpt*( ), *ptsname*( ), *unlockpt*( )

137136 XSI_FILE_SYSTEM: XSI File System
137137 *basename*( ), *dirname*( ), *lockf*( ), *mknod*( ), *nftw*( ), *seekdir*( ), *sync*( ), *telldir*( ), *utimes*( )

137138 XSI_GENERAL_TERMINAL_R: XSI Thread-Safe General Terminal
137139 *ptsname_r*( )

137140 XSI_IPC: XSI Interprocess Communication
137141 *ftok*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( ), *semctl*( ), *semget*( ), *semop*( ), *shmat*( ), *shmctl*( ),
137142 *shmdt*( ), *shmget*( )

137143 XSI_MATH: XSI Maths Library
137144 *j0*( ), *j1*( ), *jn*( ), *y0*( ), *y1*( ), *yn*( )

137145 XSI_MULTI_PROCESS: XSI Multiple Process
137146 *getpriority*( ), *getrusage*( ), *nice*( ), *setpriority*( )

137147 XSI_SIGNALS: XSI Signal
137148 *killpg*( ), *sigaltstack*( )

137149 XSI_SINGLE_PROCESS: XSI Single Process
137150 *gethostid*( ), *putenv*( )

137151 XSI_SYSTEM_DATABASE: XSI System Database
137152 *endgrent*( ), *endpwent*( ), *getgrent*( ), *getpwent*( ), *setgrent*( ), *setpwent*( )

137153 XSI_SYSTEM_LOGGING: XSI System Logging
137154 *closelog*( ), *openlog*( ), *setlogmask*( ), *syslog*( )

137155 XSI_USER_GROUPS: XSI User and Group
137156 *endutxent*( ), *getresgid*( ), *getresuid*( ), *getutxent*( ), *getutxid*( ), *getutxline*( ), *pututxline*( ),
137157 *setregid*( ), *setresgid*( ), *setresuid*( ), *setreuid*( ), *setutxent*( )

137158 XSI_WIDE_CHAR: XSI Wide-Character Library
137159 *wcswidth*( ), *wcwidth*( )

Index

IEEE SA
STANDARDS ASSOCIATION

RAISING THE WORLD'S STANDARDS

Connect with us on:

Facebook: facebook.com/ieeesa
LinkedIn: linkedin.com/groups/1791118
Beyond Standards blog: beyondstandards.ieee.org
YouTube: youtube.com/ieeesa

standards.ieee.org
Phone: +1 732 981 0060

IEEE